# RW Net 2.44

## A Network Analysis System SDK

© 2013 RouteWare / Uffe Kousgaard

# Table of Contents


Part I User Manual 11

1 Overview ... 11
2 Data sources ... 12
3 Sample code ... 14
- Point-2-Point Route ... 16
- Location-2-Location Route ... 17
- Distance Matrix ... 18
- Distance Matrix (curb approach) ... 19
- Nearest Node ... 20
- Nearest N nodes ... 21
- Nearest N locations ... 22
- TSP ... 22
- TSP - location based ... 23
- Driving Directions ... 25

4 Flowchart ... 26
5 Network terminology ... 26
6 Attributes ... 27
7 Coordinate units ... 28
8 Coordinate system ... 28
9 Password protection ... 29
10 File structure ... 29
11 Programming / GIS tools ... 30
12 DLL version ... 31
- ESRI ArcView 3 / Avenue ... 31
- MapInfo / MapBasic ... 32

13 Advanced topics ... 32
- Alpha parameter ... 32
- Hardware recommendations ... 35
- Hierarchical Routing ... 36
- Isochrones (polygon) ... 37
- Progress events ... 39
- Shortest / Fastest path ... 40
- Turn Restrictions ... 40
- Very large networks ... 41
- Limits ... 42
- Performance ... 43

14 License terms ... 44

Part II Error codes 49

Part III Check list 53


© 2013 RouteWare / Uffe Kousgaard

# Part IV Reference 57

1 AddNodes ... 57
2 AirDistNode ... 57
3 AirDistPos ... 57
4 Alpha ... 57
5 AttributeCreate ... 58
6 AttributeCreate2 ... 58
7 AttributeLoad ... 59
8 AttributeLoad2 ... 59
9 AttributeSet ... 59
10 BestNode ... 60
11 CheckLink ... 60
12 CheckNode ... 60
13 CloseLink ... 60
14 CloseLinkFixed ... 61
15 Coord ... 61
16 Coord3 ... 61
17 Coord3Cache ... 61
18 Coordinate2Location ... 62
19 Coordinate2LocationSimple ... 63
20 Coordinate2LocationIgnoreGet ... 63
21 Coordinate2LocationIgnoreNodes ... 63
22 Coordinate2LocationIgnoreReset ... 63
23 Coordinate2LocationIgnoreResetAll ... 64
24 Coordinate2LocationIgnoreSet ... 64
25 Coordinate2LocationIgnoreSetClosedLinks ... 64
26 Coordinate2Node ... 64
27 Coordinate2NodeIgnoreGet ... 65
28 Coordinate2NodeIgnoreReset ... 65
29 Coordinate2NodeIgnoreResetAll ... 65
30 Coordinate2NodeIgnoreSet ... 65
31 CoordinateExtract ... 65
32 CoordinateExtractCount ... 66
33 CoordinateWindow ... 66
34 CoordSys ... 66
35 CostDist ... 66
36 CostTime ... 67
37 Direction1 ... 67


© 2013 RouteWare / Uffe Kousgaard

38 Direction2 ..... 67
39 Directory ..... 67
40 Epsg ..... 68
41 ErrorMsg ..... 68
42 ExtendedDataCheck ..... 68
43 ExternIDfindID ..... 69
44 ExternIDfindIndex ..... 69
45 ExternIDimport ..... 69
46 ExternIDimport2 ..... 70
47 ExternIDopen ..... 70
48 ExtraDist ..... 70
49 ExtraTime ..... 70
50 ExtraVarCreate ..... 71
51 ExtraVarDrop ..... 71
52 ExtraVarExist ..... 71
53 GetLinkCost ..... 71
54 GetLinkCostDyn ..... 71
55 GetLinkCostDyn2 ..... 72
56 GetLinkDist ..... 72
57 GetLinkExtra ..... 72
58 GetLinkExtraDyn ..... 72
59 GetLinkSpeed ..... 73
60 GetLinkTime ..... 73
61 GetNodeCost ..... 73
62 GetNodeExtra ..... 73
63 GetOpenStatus ..... 74
64 GISformat ..... 74
65 Identifier ..... 75
66 Init ..... 75
67 IsoCost ..... 75
68 IsoCostDyn ..... 75
69 IsoCostDyn2 ..... 76
70 IsoCostDynLocationList ..... 76
71 IsoCostDynLocationList2 ..... 76
72 IsoCostDynLocationListN ..... 77
73 IsoCostMulti ..... 77
74 IsoCostNodeList ..... 78
75 IsoCostNodeListN ..... 78

© 2013 RouteWare / Uffe Kousgaard

76 IsoCostOffSet ..... 79
77 IsoCostSave ..... 79
78 IsoGrid ..... 80
79 IsoLink2 ..... 80
80 IsoLink2Dyn ..... 80
81 IsoLink4 ..... 81
82 IsoPoly2 ..... 81
83 IsoPoly2Fast ..... 82
84 IsoPoly2Dyn ..... 82
85 IsoPoly2DynFast ..... 83
86 IsoPoly3 ..... 83
87 IsoPoly4 ..... 84
88 LimitCreate ..... 84
89 LimitCreate2 ..... 85
90 LimitLoad ..... 86
91 LimitLoad_bitpattern ..... 86
92 Link2FromNode ..... 86
93 Link2ToNode ..... 86
94 LinkLimit ..... 87
95 LinkMax ..... 87
96 Location2Coordinate ..... 87
97 LocationListGet ..... 87
98 LocationListGetNewPos ..... 87
99 LocationListGetOldPos ..... 88
100 LocationListLimit ..... 88
101 LocationListSet ..... 88
102 Mode ..... 88
103 NearestLocation ..... 89
104 NearestNode ..... 89
105 NearestOpen ..... 90
106 NearestOpenDyn ..... 90
107 NetworkLength ..... 90
108 NodeCoordX ..... 91
109 NodeCoordY ..... 91
110 NodeCreate ..... 91
111 NodeListGet ..... 91
112 NodeListGetNewPos ..... 91
113 NodeListGetOldPos ..... 92

© 2013 RouteWare / Uffe Kousgaard

114 NodeListLimit ... 92
115 NodeListSet ... 92
116 NodeMax ... 93
117 NW3DNodes ... 93
NW3DNodesCGF ... 93
NW3DNodesMIF ... 94
NW3DNodesSHP ... 94
NW3DNodesTAB ... 94
118 NWcreate ... 94
NWcreateCGF ... 95
NWcreateMIF ... 95
NWcreateSHP ... 96
NWcreateTAB ... 96
119 NWload ... 97
120 NWloaded ... 97
121 NWunload ... 97
122 OptimumAlpha ... 98
123 POIadd ... 98
124 POIadd2 ... 98
125 POIclear ... 99
126 PositionListGet ... 99
127 PositionListSet ... 99
128 ReadSpeed ... 100
129 Result ... 100
ResultDrop ... 100
ResultGetLink ... 101
ResultGetNode ... 101
ResultSaveLink ... 101
ResultSaveNode ... 101
130 ResultFiles ... 101
ResultField ... 102
ResultFileLink ... 102
ResultFileNode ... 102
ResultFileClear ... 102
131 RoadNameCreate ... 103
132 RoadNameCreate2 ... 103
133 RoadNameLoad ... 104
134 RoadName1_Get ... 104
135 RoadName2_Get ... 104
136 RoadNameID_Get ... 105
137 RoadNameClose ... 105
138 RoundAbout ... 105
139 RoundAboutExitNode ... 105

© 2013 RouteWare / Uffe Kousgaard

140 Route ..... 106
141 RouteDyn ..... 106
142 RouteDyn_Approach ..... 107
143 RouteFileCreate ..... 108
144 RouteFileFieldAdd ..... 108
145 RouteFileRecordAdd ..... 108
146 RouteFileRecordAddCustomSection ..... 109
147 RouteFileRecordAddSection ..... 109
148 RouteFileClose ..... 109
149 RouteFind ..... 110
150 RouteFindDyn ..... 110
151 RouteGetLink ..... 110
152 RouteGetNode ..... 111
153 RouteList ..... 111
154 RouteListExitMode ..... 114
155 RouteMaxCost ..... 114
156 RouteReady ..... 115
157 RouteSave ..... 115
158 SetFastest ..... 115
159 SetLimit ..... 116
160 SetLinkResult ..... 116
161 SetLinkSpeed ..... 116
162 SetLinkSpeedDefault ..... 116
163 SetLinkTime ..... 117
164 SetNet ..... 117
165 SetShortest ..... 117
166 SetSpeed ..... 118
167 SharpTurnDrivingDirections ..... 118
168 ShowProgress ..... 118
169 ShowProgressAvenue ..... 118
170 SpatialIndexLinksPerCell ..... 119
171 StepsAdd ..... 119
172 StepsClear ..... 119
173 SwapOneWay ..... 120
174 TSP2 ..... 120
175 TSP2dyn ..... 122
176 TSP2extra ..... 122
177 TurnExport2_BIN ..... 122

© 2013 RouteWare / Uffe Kousgaard

178 TurnExport2_GIS ..... 122
179 TurnExport2_TXT ..... 123
180 TurnImport2_BIN ..... 123
181 TurnImport2_TXT ..... 123
182 TurnMandatory ..... 124
183 TurnMode ..... 124
184 TurnReset ..... 125
185 TurnRestriction ..... 125
186 TurnRestrictionComplex ..... 125
187 TurnStandard ..... 126
188 Units ..... 126
189 UTurnAllowed ..... 126
190 Valency ..... 126
191 Version ..... 127
192 ViaListClear ..... 127
193 ViaListSet ..... 127
194 ZLevelFile ..... 128

Part V Reference: Pro only 131

1 Assignment ..... 131
2 ATSP ..... 132
3 CPP ..... 132
4 CulDeSac ..... 135
5 District ..... 136
6 EncryptionKey ..... 138
7 EulerRoute ..... 139
8 FindCloseNodes ..... 139
9 FindRoundAbout ..... 140
10 Hierarchy ..... 141
11 HierarchyCheck1 ..... 141
12 HierarchyCheck2 ..... 141
13 HierarchyCreate2 ..... 142
14 HierarchyLevelSet ..... 142
15 HierarchyLoad ..... 143
16 Join ..... 143
17 Join2 ..... 144
18 NetworkCenter ..... 144
19 NetworkCenter2 ..... 145
20 Node2Link ..... 145

© 2013 RouteWare / Uffe Kousgaard

21 NodeLinkCheck ..... 146
22 NWexport ..... 146
23 Overpasses ..... 147
24 ParallelLinks ..... 147
25 RoadNameGroupBy ..... 148
26 RoadNameTest ..... 148
27 SetApproach ..... 149
28 SubNet ..... 150
29 SubNetEx ..... 150
30 UnUsedLinks ..... 151

Part VI Obsolete functions 155

1 Coord2 ..... 155
2 Direction0 ..... 155
3 FindNode ..... 155
4 FindNodeFast ..... 155
5 IsoPoly1 ..... 156
6 IsoPoly1new ..... 156
7 NodeIndexCreate ..... 157
8 NodeIndexDrop ..... 157
9 NodeIndexExist ..... 157
10 RouteVia ..... 158
11 TSP ..... 158
12 TSPextra ..... 159
13 TurnExport ..... 159
14 TurnImport ..... 159
15 VRP ..... 160
VRPcreate ..... 163
VRPmatrix ..... 163
VRPcustomer ..... 163
VRPvars ..... 164
VRPcoordinate ..... 164
VRPrunSweep ..... 164
VRPrunCut ..... 165
VRPrunGrow ..... 165
VRPresults ..... 165
VRPresultsRoute ..... 166
VRPresultsRouteIndex ..... 166
VRPdrop ..... 166

© 2013 RouteWare / Uffe Kousgaard

# Part I

# User Manual

# 1 User Manual

## 1.1 Overview

### RW Net 2.44

This documentation is best read by first stepping through most chapters in the User Manual section. The last chapters are not as important as the first, so you can skip them in the first phase (advanced topics).

The Reference Manual section contains an alphabetical list of all functions.

A good way to learn the many functions is to try RW Net Demo, which has been installed. Go to Start > Programs > RW Net > RW Net Demo Application and try it. The source code for this application is also included with RW Net (see Delphi demos).

This document is available as a PDF file from the website for easier printing and reading off-line.

#### Introduction

RW Net is a general purpose routing library. It is flexible enough to be used together with almost any GIS system 30 available and it will also work together with most programming tools 30 on the market.

RW Net uses it's own format for storing street networks and included are functions for importing 94 street databases from most common GIS formats. This topological format is targeted towards routing purposes and is described here 29. RW Net always loads 97 the topological network into memory before doing any calculations.

The basic structure in the topological network is a one-to-one relationship, where the first link in the network matches the first record in your GIS file, second link matches second record etc. This makes the network files very compact and fast to use.

All attribute 27 information (road class, one-way information etc.) is held in a separate data structure which can easily be changed without having to re-create the topological network. The 4 available modes can be used for defining streets which are impassable to heavy vehicles, temporarily closed roads etc.

#### Using RW Net classes

RW Net consists of 2 main objects: RWnetBase and RWcalc, where RWnetBase holds all network information and RWcalc does the actual calculations. RWnetBase can be used on it's own for a few tasks, but RWcalc will always have to be linked to an instance of RWnetBase. Use function SetNet 117 for linking them.

Several RWcalc can be linked to the same RWnetBase in a multi-threaded application.

The DLL version 31 is slightly different in this area.

#### Standard functions

This is a short description of some of the main function groups:

Route 106 is used for calculating the cheapest route between two locations. Cheapest can be either shortest, fastest or any combination hereof and besides this you can also have an extra variable

© 2013 RouteWare / Uffe Kousgaard

calculated along the same route. This can e.g. be used for calculating the distance along the fastest route. It is also possible to do exact routing from somewhere on one street section to somewhere on another street section - we call that dynamic segmentation (RouteDyn 106).

IsoCost 75 is used for calculating link-based isochrones, but it's also very handy for creating distance matrices in a very efficient manner. IsoCostDyn 75 is useful for the same purposes, just with dynamic segmentation.

TSP2 120 and TSP2dyn 122 is used for optimizing the sequence of visiting a list of customers.

The Coordinate2Node 64 and Coordinate2Location 62 functions transform a set of coordinates into either a node ID or a location.

Turn restrictions 40 are supported in several ways.

The ExternID 69 functions makes it possible to use a secondary ID for all link references.

Routes can be restricted to streets, that are wide enough, bridges high enough etc. This is done by setting a limit 84.

RouteList 111 makes it possible to create driving directions and many other more complex results with just a few function calls.

**RW Net Pro**

See a list 131 of additional functions.

**Output**

RW Net has several ways of returning results to your application:

- As a direct result from a function as when "Route" returns the cost of the cheapest route.
- Functions which has a more complex output (e.g. RouteFind or IsoCost), can be queried section by section
- As a GIS file (MIF, SHP, TAB, GML 2.1.2 or KML 2.0), which can be imported into your GIS afterwards.

**Updates**

RW Net is being updated all the time with bugs being corrected and new functions added. You can at any time click here to see the latest changes directly from the website. The list of changes is also your key to checking out, if you need to update your current code, when new versions are released. New versions of RW Net are generally backwards compatible, but there are a few exceptions.

## 1.2 Data sources

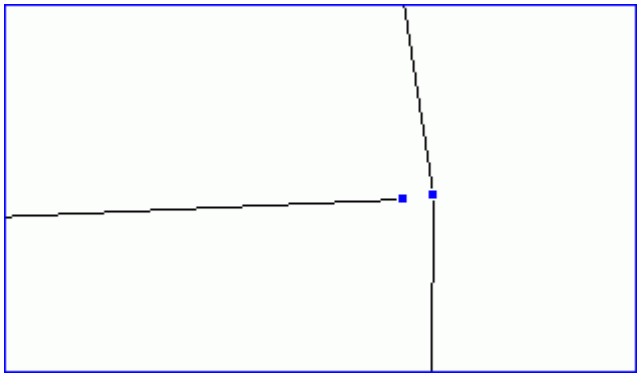
At RouteWare website you will find a list of street data providers for various parts of the world. Data from these providers usually have a topological correct structure, which means they are almost ready to use in RW NetServer.

But how should your own street data look like, in order to be used in RW NetServer? ***The short answer is they should snap and split at intersections and the network should be plane unless there is an overpass.***

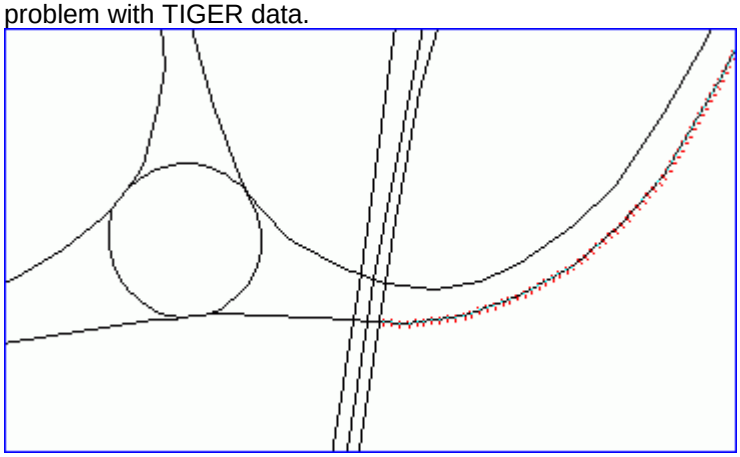
Below is shown some examples on networks, which are NOT correct, but all look correct unless you check out the details:

© 2013 RouteWare / Uffe Kousgaard

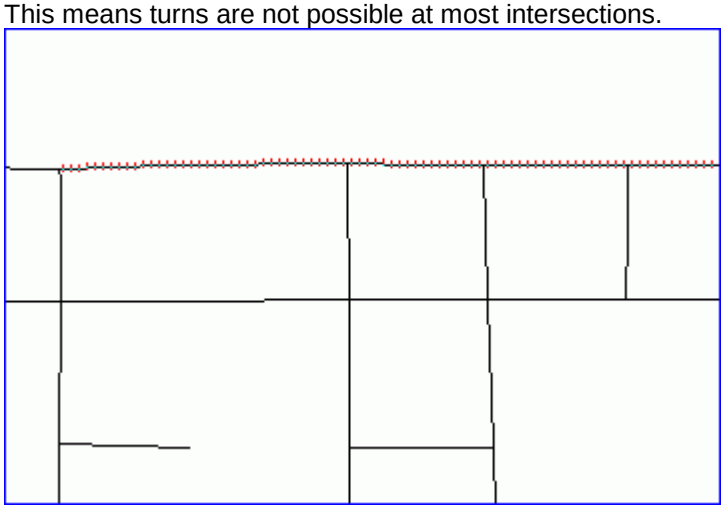
**Example 1: Missing snap at an intersection**
This means the network doesn't connect and the movement to / from the disconnected section, isn't possible. In the example below, the gap is just 1 meter and can't be seen at normal zoom levels.



**Example 2: Split at overpass / underpass**
This means a lot of impossible turn movements are suddenly made possible. This is a typical problem with TIGER data.



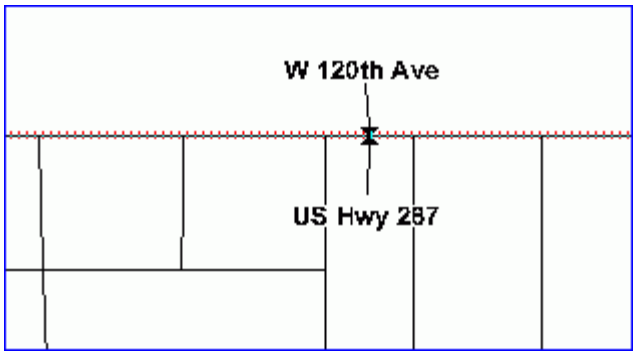
**Example 3: Doesn't split/break at intersections**
This means turns are not possible at most intersections.



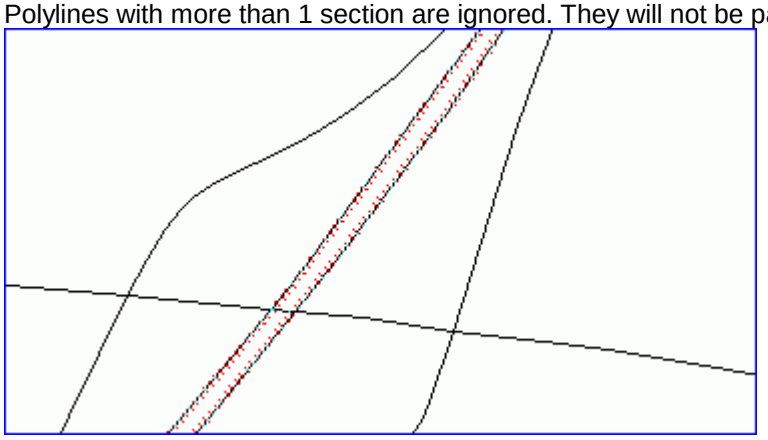
**Example 4: Double digitization with two street names, here name + route number**

© 2013 RouteWare / Uffe Kousgaard

Not a really big problem, but the result of a route calculation may include one of the two streets in a more or less random fashion.




**Example 5: Multi sectioned polylines**
Polylines with more than 1 section are ignored. They will not be part of any route.



## 1.3 Sample code

Most of the sample code provided are some very basic examples, which show how to generate a set of network files and calculate a route. It is not possible to provide sample code for all GIS and programming tools 30, but over time we are adding more.

For the most comprehensive example on how to perform various functions, you should look at RW Net Demo, which is both available as a compiled application at RouteWare website and the source code is included as part of the Delphi Sample code install.

**Step-by-step sample (no specific language)**

Below you will find a commented sample, which will show you which RW Net functions to call, to generally setup a street network for routing. All checking of error codes 49 from the functions has been omitted to make it more readable:

*Generally you should use the same order (see also here 26) in your code as shown below, or your calculations may fail !*

- **Initial setup**

This part has to be performed every time you start using RW Net

*Enter the password 29 (not needed in VCL, Free and time-limited versions)*
`RWnetbase1.init 75("yourpassword")`

*Link the calculation object to the network object*

© 2013 RouteWare / Uffe Kousgaard

```
OCX/.NET: RWcalc1.SetNet(rwnetbase1.identifier)
VCL: RWcalc1.SetNet(rwnetbase1)
DLL: Not needed
```

*Choose a coordinate unit*
RWnetBase1.Coord = LatLong

*Choose a GIS output format - we pick SHP*
RWnetBase1.GISformat = gf_SHP

- **Creating the network**

This part is only needed the first time or when your base street network changes. This is also the only functionality in RW Net, which can **not** be distributed royalty free.

*Actual creation of binary network from a SHP file called streets.shp, streets.shx*
*Second parameter is true, so the node layer is created for visual inspection*
*Third parameter means a default coordinate system is generated*
`RWnetBase1.NWCreateSHP("streets",true,"")`

Eventually set property ZlevelFile first.

- **Creating attributes**

This part is only needed the first time or when the attributes of your base street network changes

*First a field in the DBF file should be prepared according to the description here*
*Actual creation of attributes from DBF file*
`RWnetBase1.AttributeCreate2("streets.dbf","attrib",0)`

- **Creating roadnames**

This part is only needed the first time or when the roadnames of your base street network changes. It is also only needed, if you want to create driving directions or use one of the other functions, that work with roadnames.

*It is possibly to read directly from a DBF file:*
```
RWnetBase1.RoadNameCreate2(1,"streets.dbf","name","",0,0)
```
(see RoadNameCreate for an alternative method)

- **Loading the network**

*Define the units, when distances are reported. This can be changed at any time.*
*This setting also applies to the SetSpeed command later*
`RWnetBase1.Units = miles`

*We don't need support for turn restrictions. Set this before NWload()*
`RWnetBase.TurnMode = false`

*The network is loaded into RAM*
`RWnetBase1.NWLoad()`

*We define the speed (km/h or miles/h) for every road class (1..7) in the dataset. This has to be before AttributeLoad*
```
RWnetBase1.SetSpeed(1,1)
RWnetBase1.SetSpeed(2,5)
RWnetBase1.SetSpeed(3,20)
RWnetBase1.SetSpeed(4,25)
RWnetBase1.SetSpeed(5,35)
RWnetBase1.SetSpeed(6,45)
```

© 2013 RouteWare / Uffe Kousgaard

```
RWnetBase1.SetSpeed(7,55)
```

*We load attribute information and define that we want to apply the one-way information.*
```
RWnetBase1.AttributeLoad(true)
```

*We load roadnames from file 1, but don't cache the actual names (only required for sample 9 below)*
```
RWnetBase1.RoadNameLoad(1,false)
```

*Define fastest path*
```
RWcalc1.SetFastest()
```

*Set alpha to the best possible value. Has to be after AttributeLoad and SetFastest.*
```
RWcalc1.Alpha = RWcalc1.OptimumAlpha()
```

*Include the extra variable which can now hold distance (for the fastest path)*
```
RWcalc1.ExtraVarCreate()
```

Turn restrictions can also be loaded by calling RWnetBase1.TurnImport2_TXT.

Now that the network has been loaded etc. it is time to do some actual calculations:

1) A route
2) A route - dynamic segmentation
3a) A distance matrix
3b) A distance matrix (Curb Approach)
4) Nearest node in one theme for all nodes in another theme
5) Nearest N nodes in one theme for all nodes in another theme
6) Nearest N locations in one theme for all locations in another theme
7) TSP - travelling salesman
8) TSP - travelling salesman - dynamic segmentation
9) Driving directions

### 1.3.1 Point-2-Point Route

(continued from click here)

This is an example, which calculates a route and then stores the route as SHP file at the end, which can be visualized.

**NOTE: From v2.24 this can also be done with function RouteList instead - see sample.**

*Native coordinates of from and to positions*
```
x1 = 100.12
y1 = 389.23
x2 = 193.49
y2 = 331.73
```

*Calculate nearest start/end nodes*
```
RWnetBase1.Coordinate2node(x1,y1,node1,dist1)
RWnetBase1.Coordinate2node(x2,y2,node2,dist2)
```

*Actual route calculation - returns time (always minutes) and distance*
```
routetime = RWcalc1.Route(node1,node2)
routedist = RWcalc1.GetNodeExtra(node2)
```

*Scan the route to create the route as "output.shp"*

© 2013 RouteWare / Uffe Kousgaard

```
RWcalc1.RouteFileCreate("output",1)
RWcalc1.RouteFileFieldAdd("linkID",2,0,0)
for i = RWcalc1.RouteFind(node2)-1 to 1 step -1
  t = rwcalc1.RouteGetLink(i)
  RWcalc1.RouteFileRecordAdd(1,str$(abs(t)))
  if t>0 then
    rwcalc1.RouteFileRecordAddSection(t,0,1)
  else
    rwcalc1.RouteFileRecordAddSection(-t,1,0)
  end if
next i
RWcalc1.RouteFileClose
```

Now you can show the route on a map together with the base street network and the route time & distance.

### 1.3.2 Location-2-Location Route

(continued from click here 14)

This is an example, which calculates a route with dynamic segmentation and then stores the route as SHP file at the end, which can be visualized.

**NOTE: From v2.24 this can also be done with function RouteList instead - see sample 25.**

*Native coordinates of from and to positions*

```
x1 = 100.12
y1 = 389.23
x2 = 193.49
y2 = 331.73
```

*Calculate nearest start/end locations*

```
RWnetBase1.Coordinate2location(x1,y1,link1,percent1,side,dist,xnew,ynew)
RWnetBase1.Coordinate2location(x2,y2,link2,percent2,side,dist,xnew,ynew)
```

*The routedyn function doesn't accept 0 or 1 as percentage, so we make this check first:*

```
if percent1=0 then percent1 = 0.0001 else if percent1=1 then
percent1=0.9999 end if
if percent2=0 then percent2 = 0.0001 else if percent2=1 then
percent2=0.9999 end if
```

*Actual route calculation - returns time (always minutes) and distance*

```
routetime = RWcalc1.RouteDyn
(link1,link2,Percent1,Percent2,fromto1,fromto2,routelength,routedist)
```

(See function RouteFindDyn 110 for further details for possible performance improvements)

*Scan the route to create the route as "output.shp"*

```
rwcalc1.RouteFileCreate("output",1)
rwcalc1.RouteFileFieldAdd("ID",2,0,0)
rwcalc1.RouteFileRecordAdd(1,"1")
if (link1=link2) and (routelength=0) then
  rwcalc1.RouteFileRecordAddSection(link1,percent1,percent2)
else
  if fromto1=0 then
    rwcalc1.RouteFileRecordAddSection(link1,percent1,0)
  else
    rwcalc1.RouteFileRecordAddSection(link1,percent1,1)
```

© 2013 RouteWare / Uffe Kousgaard

```
  end if
  for i = routelength-1 to 1 step -1
    rwcalc1.RouteFileRecordAdd(1,str$(routelength+1-i))
    t = rwcalc1.RouteGetLink(i)
    if t>0 then
      rwcalc1.RouteFileRecordAddSection(t,0,1)
    else
      rwcalc1.RouteFileRecordAddSection(-t,1,0)
    end if
  next i
  rwcalc1.RouteFileRecordAdd(1,str$(1+routelength))
  if fromto2=0 then
    rwcalc1.RouteFileRecordAddSection(link2,0,percent2)
  else
    rwcalc1.RouteFileRecordAddSection(link2,1,percent2)
  end if
end if
RWcalc1.RouteFileClose
```

Now you can show the route on a map together with the base street network and the route time & distance.

### 1.3.3 Distance Matrix

(continued from click here 14)

Distance matrices are a very important part of street network calculations, as it is usual input to many other functions. In this example, it is assumed that we have two themes of point-objects and we want to calculate the distance from all elements in the first theme to all elements in the second theme. The two themes may be the same.

For simplicity, we assume access to the coordinates of the point objects in 2 arrays (x1,y1 for theme1 and x2,y2 for theme2):

*This first loop calculates the nearest node for all points in theme2, so we don't have to do that for every time in the inner loop:*

```
for j = 1 to theme2.count
  RWnetBase1.Coordinate2node(x2[j],y2[j],node2[j],dist)
  RWcalc1.nodelistset(j,node2[j])
next j
```

*Main loop*

```
for i = 1 to theme1.count
  RWnetBase1.Coordinate2node(x1[i],y1[i],node1,dist)
  RWcalc1.isocost(node1,0) or RWcalc1.isocostnodelist(node1,theme2.count)
  for j = 1 to theme2.count
    routetime = RWcalc1.GetNodeCost(node2[j])
    if routetime<1e36 then
      routedist = RWcalc1.GetNodeExtra(node2[j])
      print "From "+str$(i)+" (theme1) to "+str$(j)+" (theme2):
"+str$(routetime)+" minutes and "+str$(routedist)+" miles"
    else
      print "From "+str$(i)+" (theme1) to "+str$(j)+" (theme2) : The 2
nodes are not connected"
    end if
  next j
next i
```

If the points in theme2 is located in a smaller part of the street network, the calculations can be

© 2013 RouteWare / Uffe Kousgaard

made faster, if we use the nodelist for storing theme2. This is marked with red in the sample code above. That approach requires you to have less nodes than can be in the nodelist. If your nodes are spread all over the network, it may even be a little slower (but not much).

Please also note, we don't call the RouteFind 110 function, since we are only interested in the distances and the actual list of links making up the route isn't relevant.

- **Dynamic segmentation**

If you want to calculate a distance table based on dynamic segmentation instead, it is possible to use the same fast isochrone approach as above:

```
for j = 1 to theme2.count
  RWnetBase1.Coordinate2location(x2[j],y2[j],link2[j],percent2
[j],side,dist,x,y)
  RWcalc1.locationlistset(j,link2[j],percent2[j])
next j

for i = 1 to theme1.count
  RWnetBase1.Coordinate2location(x1[j],y1
[j],link1,percent1,side,dist,x,y)
  RWcalc1.isocostdynlocationlist(link1,percent1,theme2.count)
  for j = 1 to theme2.count
    routetime = RWcalc1.getlinkcostdyn(link2[j],percent2[j])
    if routetime>=0 then
      routedist = RWcalc1.getlinkextradyn(link2[j],percent2[j])
      print "From "+str$(i)+" (theme1) to "+str$(j)+" (theme2):
"+str$(routetime)+" minutes and "+str$(routedist)+" miles"
    else
      print "From "+str$(i)+" (theme1) to "+str$(j)+" (theme2) : The 2
locations are not connected"
    end if
  next j
next i
```

### 1.3.4 Distance Matrix (curb approach)

(continued from click here 14)

This example is mostly for those that need a distance matrix for further calculations, where curb approach matters.

Make sure turnmode 124 is true.

*Set up N random locations, where link number is from 1 to 1000*

```
for i = 1 to N
  rwcalc1.LocationListSet(i,trunc(random*1000)+1,0.5)
next i
```

*Handle combinations where link1 <> link2*

```
for approach1 = 1 to 2
  for i = 1 to N
    rwcalc1.LocationListGet(i,link1,percent1)
    rwcalc1.IsoCostDynLocationList2(link1,percent1,N,approach1*512)
    for j = 1 to N
      rwcalc1.LocationListGet(j,link2,percent2)
      if link1<>link2 then
        for approach2 = 1 to 2
          matrix[approach1,approach2,i,j] =  rwcalc1.GetLinkCostDyn2
```

© 2013 RouteWare / Uffe Kousgaard

```
(link2,percent2,extra,approach2*512)
        next approach2
      end if
    next j
  next i
next approach1

Handle combinations where link1 = link2
for i = 1 to N
  rwcalc1.LocationListGet(i,link1,percent1)
  for j = 1 to N
    rwcalc1.LocationListGet(j,link2,percent2)
    if link1=link2 then
      for approach1 = 1 to 2
        for approach2 = 1 to 2
          c = rwcalc1.RouteDyn_Approach(link1,link2,percent1,percent2,
                                        approach1*512,approach2*512,
                                        fromto1,fromto2,rlength,extra)
          if c<0 then
            matrix[approach1,approach2,i,j] = 1e38
          else
            matrix[approach1,approach2,i,j] = c
          end if
        next approach2
      next approach1
    end if
  next j
next i
```

It is important to do it as 2 loops, or the calls to RouteDyn_Approach will clear the isochrone setup previously.

### 1.3.5 Nearest Node

(continued from click here 14)

When you want to know the nearest object in a store theme for every object in a customer theme, there is generally no faster way to do it, but calling the function NearestNode like this:

For simplicity, we assume access to the coordinates of the point objects in 2 arrays (x1,y1 for customer theme and x2,y2 for store theme):

```
for j = 1 to storetheme.count
  RWnetBase1.Coordinate2node(x2[j],y2[j],node2,dist)
  RWcalc1.nodelistset(j,node2)
next j

for i = 1 to customertheme.count
  RWnetBase1.Coordinate2node(x1[i],y1[i],node1,dist)
  ID = RWcalc1.nearestnode(node1,storetheme.count)
  if ID > 0 then
    node2 = RWcalc1.nodelistget(ID)
    routetime = RWcalc1.GetNodeCost(node2)
    routedist = RWcalc1.GetNodeExtra(node2)
    print "Nearest for customer "+str$(i)+": Store "+str$(ID)+" was
nearest and "+str$(routetime)+" minutes and "+str$(routedist)+" miles
away"
  else
    print "Nothing found for customer "+str$(i)
  end if
```

© 2013 RouteWare / Uffe Kousgaard

```
next i
```

If store theme has more nodes than can be in the node list, you will have to calculate an isochrone so big that at least one node is within the isochrone and then traverse all nodes in store theme to find the nearest - this will be slower and if your isochrone doesn't contain any nodes at all, you will have to increase the size of the isochrone.

- **Faster method**

In the example above, we have calculated the distance from the customer theme to storetheme. In many situations we don't really care if we use the other direction (from storetheme to customer theme), since we are only interested in the total distance anyway. To make it as correct as possible (even if we have one-way restrictions), we can swap the direction of all restrictions first by using the SwapOneWay function. Note: There is no function for "swapping" the direction of turn-restrictions !

Now, we can use this method instead:

```
for j = 1 to storetheme.count
  RWnetBase1.Coordinate2node(x2[j],y2[j],node2,dist)
  RWcalc1.nodelistset(j,node2)
next j
```

*The key to get fast results is to choose the 125 minutes as close to the maximum time from any customer to a store as possible*

```
RWnetBase1.SwapOneWay()
RWcalc1.isocostmulti(storetheme.count,125)
RWnetBase1.SwapOneWay()

for i = 1 to customertheme.count
  RWnetBase1.Coordinate2node(x1[j],y1[j],node1,dist)
  routetime = RWcalc1.GetNodeCost(node1)
  if routetime<1e36 then
    routedist = RWcalc1.GetNodeExtra(node1)
    print "Nearest for customer "+str$(i)+": Store "+str$(ResultGetNode
(node1))+" was nearest and "+str$(routetime)+" minutes and
"+str$(routedist)+" miles away"
  else
    print "Nothing found for customer "+str$(i)
  end if
next i
```

### 1.3.6 Nearest N nodes

(continued from click here 14)

When you want to know the nearest N objects in a store-theme for every object in a customer-theme, you should use function isocostnodelistN:

For simplicity, we assume access to the coordinates of the point objects in 2 arrays (x1,y1 for customer-theme and x2,y2 for store-theme):

*Enter stores into node list*

```
for j = 1 to storetheme.count
  RWnetBase1.Coordinate2node(x2[j],y2[j],node2,dist)
  RWcalc1.nodelistset(j,node2)
next j

for i = 1 to customertheme.count
```

© 2013 RouteWare / Uffe Kousgaard

```
  RWnetBase1.Coordinate2node(x1[i],y1[i],node1,dist)
Locate 5 nearest objects
  found = RWcalc1.isocostnodelistN(node1,storetheme.count,5,0)
  if found > 0 then
    for j = 1 to found
      index = rwcalc1.NodeListGetNewPos(j);
      node = rwcalc1.nodelistget(index);
      print "Customer "+str$(i)+": "+str$(j)+": "+str$(node)+"
"+str$(rwcalc1.getnodecost(node))
    next
  else
    print "Nothing found for customer "+str$(i)
  end if
  print " "
next i
```

### 1.3.7 Nearest N locations

(continued from click here 14)

When you want to know the nearest N objects in a store-theme for every object in a customer-theme, you should use function IsoCostDynLocationListN:

For simplicity, we assume access to the coordinates of the point objects in 2 arrays (x1,y1 for customer-theme and x2,y2 for store-theme):

```
Enter stores into location list
for j = 1 to storetheme.count
  RWnetBase1.Coordinate2location(x2[j],y2
[j],link2,percent2,side,dist,xnew,ynew)
  RWcalc1.locationlistset(j,link2,percent2)
next j

for i = 1 to customertheme.count
  RWnetBase1.Coordinate2location(x1[i],y1
[i],link1,percent1,side,dist,xnew,ynew)
Locate 5 nearest objects
  found = RWcalc1.isocostdynlocationlistN
(link1,percent1,storetheme.count,5,0)
  if found > 0 then
    for j = 1 to found
      index = rwcalc1.LocationListGetNewPos(j);
      rwcalc1.locationlistget(index,link2,percent2);
      print "Customer "+str$(i)+": "+str$(j)+":
"+str$(rwcalc1.getlinkcostdyn(link2,percent2))
    next
  else
    print "Nothing found for customer "+str$(i)
  end if
  print " "
next i
```

### 1.3.8 TSP

(continued from click here 14)

This shows how to calculate a route, which passes through several nodes in an optimum order. The result of the route is stored in an external file.

© 2013 RouteWare / Uffe Kousgaard

**NOTE: From v2.24 this can also be done with function RouteList instead - see sample 25.**

For simplicity, we assume access to the coordinates of the point objects in an array:

```
This first loop enters all points into the node list
for i = 1 to theme.count
  RWnetBase1.Coordinate2node(x[i],y[i],node,dist)
  RWcalc1.nodelistset(i,node)
next i

Enable ATSP mode (only available in RW Net Pro)
RWcalc1.ATSP = true

Actual TSP calculation (mode = 1, meaning round-trip and no time-limit on calculations)
routetime = RWcalc1.TSP2(theme.count,1,0)
routedist = RWcalc1.TSP2extra

Create the route as "output.shp"
RWcalc1.RouteFileCreate("output",2)
RWcalc1.RouteFileFieldAdd("LinkID",2,0,0)
RWcalc1.RouteFileFieldAdd("Minutes",5,0,0)
TotalTime = 0
for j = 1 to theme.count  (in mode 0 or 2, use count-1)
  node1 = RWcalc1.nodelistget(RWcalc1.nodelistgetnewpos(j))
  if j<theme.count then
    node2 = RWcalc1.nodelistget(RWcalc1.nodelistgetnewpos(j+1))
  else
    node2 = RWcalc1.nodelistget(RWcalc1.nodelistgetnewpos(1))
  end if

  Piecewise route calculation
  RWcalc1.Route(node1,node2)
  RWcalc1.GetNodeExtra(node2)

  for i = RWcalc1.RouteFind(node2)-1 to 1 step -1
    t = rwcalc1.RouteGetLink(i)
    Time = TotalTime + rwcalc1.GetNodeCost(rwcalc1.RouteGetNode(i))
    RWcalc1.RouteFileRecordAdd(1,str$(abs(t))+", "+str$(Time))
    if t>0 then
      rwcalc1.RouteFileRecordAddSection(t,0,1)
    else
      rwcalc1.RouteFileRecordAddSection(-t,1,0)
    end if
  next i
  TotalTime = Time
next j
RWcalc1.RouteFileClose
```

### 1.3.9 TSP - location based

(continued from click here 14)

This shows how to calculate a route, which passes through several locations in an optimum order. The result of the route is stored in an external file.

**NOTE: From v2.24 this can also be done with function RouteList instead - see sample 25.**

For simplicity, we assume access to the coordinates of the point objects in an array:

© 2013 RouteWare / Uffe Kousgaard

*This first loop enters all points into the location list*

```
for i = 1 to theme.count
  RWnetBase1.Coordinate2location(x[i],y
[i],link,percent,side,dist,xnew,ynew)
  if percent=0 then
    percent = 0.0001
  else if percent=1 then
    percent = 0.9999
  end if
  RWcalc1.locationlistset(i,link,percent)
next i
```

*Enable ATSP mode (only available in RW Net Pro)*

```
RWcalc1.ATSP = true
```

*Actual TSP calculation (mode = 0, meaning start-end trip and no time-limit on calculations)*

```
routetime = RWcalc1.TSP2dyn(theme.count,0,0)
routedist = RWcalc1.TSP2extra
```

*Create the route as "output.shp"*

```
RWcalc1.RouteFileCreate("output",1)
RWcalc1.RouteFileFieldAdd("LinkID",2,0,0)
TotalTime = 0
for j = 1 to theme.count-1 (in mode 1, use theme.count instead)
  RWcalc1.locationlistget(RWcalc1.locationlistgetnewpos
(j),link1,percent1)
  if j<theme.count then
    RWcalc1.locationlistget(RWcalc1.locationlistgetnewpos(j
+1),link2,percent2)
  else
    RWcalc1.locationlistget(RWcalc1.locationlistgetnewpos
(1),link2,percent2)
  end if
```

*Piecewise route calculation*

```
  routetime = RWcalc1.RouteDyn
(link1,link2,Percent1,Percent2,fromto1,fromto2,routelength,routedist)

  rwcalc1.RouteFileRecordAdd(1,str$(link1))
  if (link1=link2) and (routelength=0) then
    rwcalc1.RouteFileRecordAddSection(link1,percent1,percent2)
  else
    if fromto1=0 then
      rwcalc1.RouteFileRecordAddSection(link1,percent1,0)
    else
      rwcalc1.RouteFileRecordAddSection(link1,percent1,1)
    end if
    for i = routelength-1 to 1 step -1
      t = rwcalc1.RouteGetLink(i)
      rwcalc1.RouteFileRecordAdd(1,str$(abs(t)))
      if t>0 then
        rwcalc1.RouteFileRecordAddSection(t,0,1)
      else
        rwcalc1.RouteFileRecordAddSection(-t,1,0)
      end if
    next i
    rwcalc1.RouteFileRecordAdd(1,str$(link2))
    if fromto2=0 then
      rwcalc1.RouteFileRecordAddSection(link2,0,percent2)
```

© 2013 RouteWare / Uffe Kousgaard

```
    else
      rwcalc1.RouteFileRecordAddSection(link2,1,percent2)
    end if
  end if
next j
RWcalc1.RouteFileClose
```

### 1.3.10 Driving Directions

(continued from click here 14)

This shows how to calculate a route, which passes through several locations in an optimum order. The result of the route is stored in an external file with driving directions:

*Enable ATSP mode (only available in RW Net Pro)*

```
RWcalc1.ATSP = true
```

*Define 5 positions*

```
rwcalc1.PositionListSet(1,50,100)
rwcalc1.PositionListSet(2,80,70)
rwcalc1.PositionListSet(3,120,110)
rwcalc1.PositionListSet(4,70,90)
rwcalc1.PositionListSet(5,60,85)
```

*Define names and waiting time for the 5 positions*

```
rwcalc1.ViaListSet(1,"Start",0)
rwcalc1.ViaListSet(2,"Via",5)
rwcalc1.ViaListSet(3,"Via",5)
rwcalc1.ViaListSet(4,"Via",5)
rwcalc1.ViaListSet(5,"Goal",0)
```

*tspmode = 0 (optimization of via-points)*
*listmode = 3 (input via positionlist)*
*listnum = 5 (5 positions on list)*
*concatmode = 3 (driving directions mode)*
*Offroadspeed = 4 km/h*
*Starttime = 0.75 (18:00 = 6pm)*
*Timeformat = 1 (24 hour)*
*Vialist = true*
*RoadnameID = 1*

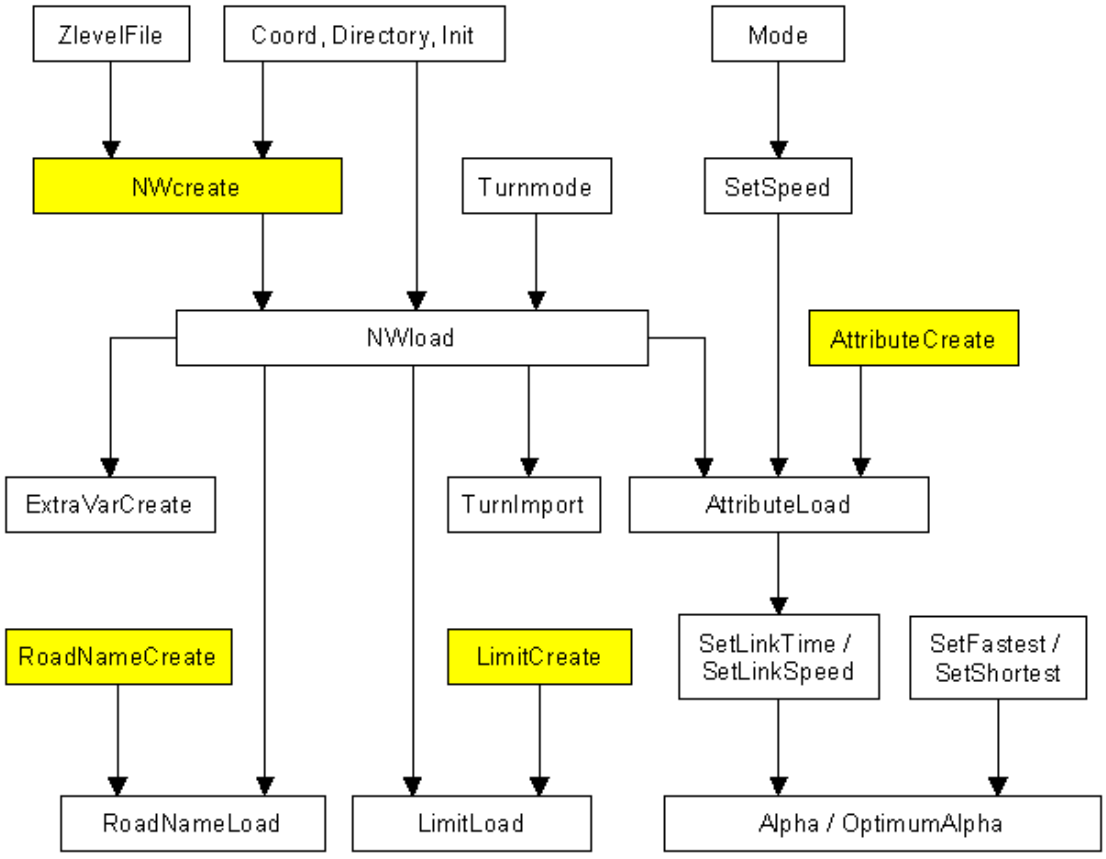
```
rwcalc1.RouteList("output",0,3,5,3,4,0.75,1,true,1,distance,time)
```

File output.shp now holds the result.

© 2013 RouteWare / Uffe Kousgaard

## 1.4 Flowchart

It is important the the functions are called in the correct order. There are more properties than the ones shown here, but these need to be set in this order:




If for instance mode is changed, you will have to call setspeed, attributeload, reapply local changes to Linktime or Linkspeed and recalculate alpha.

You can alse see, that turnmode can not be changed without reloading the network.

## 1.5 Network terminology

Terminology used to describe the various elements of a street network:

A *link* consists of several connected vertices (2 or more, blue squares on the map below). The first vertex of a link is called the *start-node / from-node* and the last vertex is called the *end-node / to-node*. See also function Link2FromNode 86 and Link2ToNode 86.

Most of the nodes share coordinates with nodes of other links. The number of links sharing a node is referred to as the *valency* of the node. You should normally never reach more than 10. See also function Valency 126.

A node is also called an *intersection* - even if the valency is 1 or 2. A link where one of the nodes has valency 1 is called a *dangling* link. A node with valency 2 is called a *pseudo-node* (see also function Join 143). The red node on the map is such a pseudo-node.

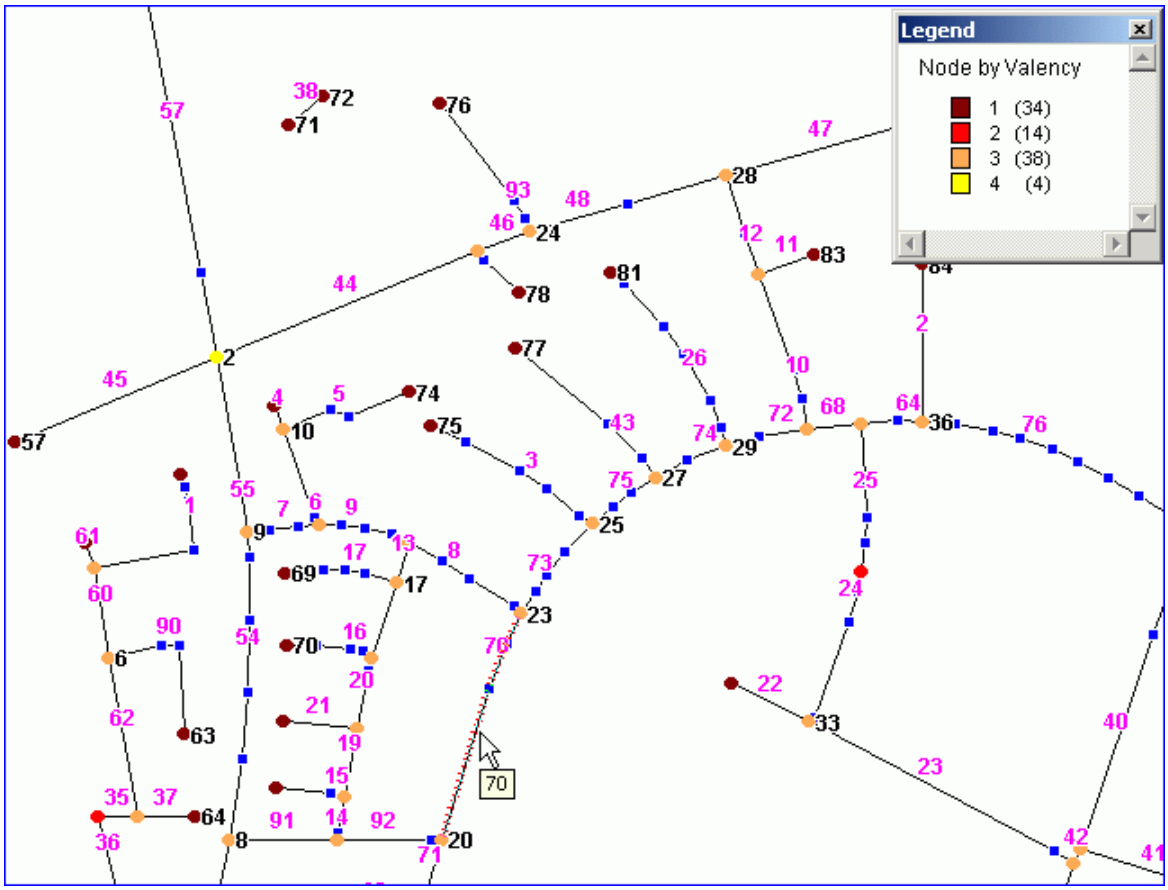
A link is identified by it's ID (Magenta text on the map: 1, 2, 3 ....), which corresponds to the record

© 2013 RouteWare / Uffe Kousgaard

ID's of the input dataset used by function NWcreate.

A node is identified by it's ID (Black text on the map: 1, 2, 3 ....). Node ID's are primarily ordered by valency in descending order and secondarily by x-coordinate in ascending order. Node ID's are assigned during network creation and can not be controlled by the user.

A *location* is a position on a link: e.g. 50% along link 70 - counted in the same direction as that the link has been digitized in. This app. matches the cursor on the map below. Locations are used when doing dynamic routing. The percentage needs to be strictly *between* 0 and 1.




## 1.6 Attributes

The attribute for each link in the network play a key role in defining how the link is used in the routing calculations. This is defined through a bit-pattern:

0-31: Defines road class. These have no predefined meaning, but their value is translated into a drivetime during attributeload 59 by looking up the value in the array of speeds 118.

Add 32 if mode 1 isn't allowed on this link.
Add 64 if mode 2 isn't allowed on this link.
Add 128 if mode 3 isn't allowed on this link.
Add 256 if mode 4 isn't allowed on this link.
Add 512 if it is a one-way street, which may not be travelled in the opposite of the digitised direction.
Add 1024 if it is a one-way street, which may not be travelled in the digitised direction.
Add 2048 if a link is part of a roundabout. This is used for generating driving directions. See RouteList 111.

© 2013 RouteWare / Uffe Kousgaard

Add 4096 if a link is a "non-driving" link. This can be used to mark ferries and car trains, so their length is not included in driving directions. See RouteList 111.
Add 8192 if it is not allowed to make U-turns at the From-end of the link.
Add 16384 if it is not allowed to make U-turns at the To-end of the link.

32768 is reserved for future use.

An example:

A road of class 4, which can only be travelled in the direction of digitization: 4 + 512 = 516.

## 1.7 Coordinate units

2 primary kind of coordinate units are supported:

- Spheric / Latitude-longitude
- Cartesian / Projected

When working with spheric coordinates, all distance calculations are performed using great circle distances and the Earth is considered a perfect sphere with radius 6378130 meter.

When working with cartesian coordinates, all distance calculations are performed using straight Pythagoras formula. Several different cartesian units are supported. See property Coord 61 for a full list.

It is worth noting, that RW Net *never* performs any transformation between coordinate systems. It always works with the native coordinates of the base dataset used when creating the network. It will return strange results, if you set the coordinates as spheric, while they are really meters or vice-versa. It is YOUR responsibility to make sure this is correct.

See also Coordinate system 28 and Units 126.

## 1.8 Coordinate system

When generating a binary network from a GIS street database, information about the coordinate system is stored inside the binary files 29. This is stored in the same format as used by MapInfo coordinate clauses (text format). The maximum length of the text is 240 characters.

When generating the binary network from TAB or MIF files, this is done automatically. When generating it from SHP or CGF files, you have the option of passing the string as a parameter. If an empty string is passed, a default coordinate clause is generated.

You can at any time get the coordinate clause of the currently loaded network from property CoordSys 66.

The coordinate clause is used, when native GIS files are generated as output from various RW Net functions (e.g. NodeCreate 91 etc.).

**TAB / MIF format**
When the result of a calculation is stored as a GIS file in either TAB or MIF format, the coordinate clause is used to generate output with the same coordinate system as the input GIS street database. In order to create TAB files, you will need MITAB.DLL installed on your system. Due to the status of the TAB file format (no public, official description), it will sometimes be possible that you encounter a TAB file or coordinate clause, which isn't handled properly by either function NWcreateTAB 96 or MITAB library. At the moment only TAB files with custom affine coordinate systems are known to not be supported by NWcreateTAB 96.

© 2013 RouteWare / Uffe Kousgaard

**SHP format**
If the chosen output format is SHP, a PRJ file is not created. It is the responsibility of the programmer to copy the PRJ file from the original SHP street database to the new SHP file, if needed.

**CGF format**
This has no concept of coordinate systems.

## 1.9 Password protection

When you use RW Net DLL, OCX or .NET assembly, Standard / Pro version, you will have to use a password to unlock the functionality. Call function Init 75 for this before calling any other functions. If a proper password isn't entered, RW Net will crash on the first call of a function.

RW Net Free and Time-limited demo versions don't need a password.

## 1.10 File structure

When a network is created 94 from a GIS file (SHP, TAB, MIF or CGF format), several binary files are created on disk, which together define the topological network. This gives a short description of the content of the various files:

| Filename | Mandatory | Encrypted | Explanation |
|---|---|---|---|
| Attribute.bin | | X | Attributes of links |
| Coord.bin | X | X | Coordinates of all nodes |
| Coord3.bin | | X | Coordinates of the rest of the vertices |
| Dist.bin | X | X | Length of all links in the network |
| Hierarchy.bin | | X | Hierarchy attribute (new from v. 2.29) |
| Index.bin | | | Index for conversion between link id (1, 2, 3..) and external id. |
| Limit?.bin | | X | Information about limits on links such as max heights etc. (new from v. 2.19) |
| Link.bin | X | X | Information about link-node relationship ("topology") |
| Node.bin | X | X | Information about link-node relationship ("topology") and coordinate clause 28 |
| Roadname??.bin | | | List of possible road names (new from v.2.24) |
| Roadnumber??.bin | | X | Index into roadname.bin (new from v.2.24) |
| Spatialindex.bin | | | Spatial index of both links and nodes (new from v. 2.17) |
| Turn?.bin | | | Turn restrictions |
| Turn2_?.bin | | | Turn restrictions (new from v. 2.33) |

If you drop coord3.bin, these functions will not be possible to call:
- AddNodes 57 property <> 0
- Coordinate2location 62
- Location2coordinate 87
- IsoLink2 80
- IsoLink4 81
- All RouteFile 108 functions
- Direction2 67
- NWexport 146
- TurnStandard 126

© 2013 RouteWare / Uffe Kousgaard

Limit.bin files are required to implement functions such as limiting a route to sections of sufficient width, weight etc.

If you set the EncryptionKey 138 property, files marked as such in the table will be encrypted during creation and decrypted during load.

If you drop spatialindex.bin, these function will not be possible to call:
- Coordinate2location 62
- Coordinate2node 64
- Overpasses 147

See also NWload 97, NWcreate 94, AttributeCreate2 58, LimitCreate 84, HierarchyCreate2 142 and RoadNameCreate2 103.

*File coord2.bin is obsolete and should be replaced by coord3.bin*

## 1.11 Programming / GIS tools

RW Net is available for different programming languages:

**As VCL components:**
- Borland Delphi 5-7, 2005-2006
- CodeGear RAD Studio 2007-2009-2010 (Delphi)
- Embarcadero RAD Studio XE (Delphi)

**As OCX / Active-X component:**
- MS Visual Basic 6.0
- MS Visual C++ 6.0
- Other OCX compliant languages

**As 32-bit DLL:**
- ESRI ArcView 3 (Avenue)
- MapInfo MapBasic

**As .NET 1.1 assembly:**
- MS Visual Studio .NET 2003
- Borland Developer Studio 2006 (C# and Delphi.NET)
- Other .NET languages

**As .NET 2.0 assembly:**
- MS Visual Studio .NET 2005 / 2008
- CodeGear RAD Studio 2007 (Delphi.NET)
- Other .NET languages

**As Win CE component (Pro only)**
- Free Pascal 2.4.0, ARM compiler

It is of course possible to develop applications without any visual map at all, if you don't need to "see" the results, but here is a list of some GIS tools possible for use with RW Net, if you want to develop mapping applications:

**Standalone Win32 applications:**
Delphi / C++ Builder / Kylix: CartoVCL, TatukGIS
OCX: CartoX, GeoObjects, MapObjects, MapObjects LT, MapWinGIS, MapX, TatukGIS

© 2013 RouteWare / Uffe Kousgaard

**.NET / Web applications:**
Browse-A-Map, MapDotNet, MapXtreme, TatukGIS

**Integrated mapping:**
ArcGIS, ArcView 3, MapInfo

**Map servers for the internet / intranet:**
ArcGIS Server, AspMap, MapGuide, MapServer, MapXtreme, TatukGIS

## 1.12 DLL version

When you use the DLL version of RW Net, there are a few differences compared to the VCL / OCX version:

The DLL version will only allow one network to be loaded at a time because only 1 instance of the network and calculation object is auto-created, when the DLL is initialized. This also means multi-threaded applications are not possible.

You also don't have to mess with both rwnetbase and rwcalc objects. All functions are called directly.

The DLL has built-in dialogs for showing progress 39, when various functions are called.

RW Net DLL is only supplied with headers for Avenue 31 and MapBasic 32. If you want to use it with other programming languages, which don't support OCX'es, you will have to translate the headers yourself. The header for MapBasic is then probably the easiest starting point.

### 1.12.1 ESRI ArcView 3 / Avenue

To use RW Net with Avenue programming, you should use the DLL version of RW Net. The sample project (sample1.apr) already has header definitions for all functions, so you can see how to call the various functions from within Avenue.

Since Avenue doesn't support double precision when calling DLL's, all functions with double precision variables, comes in 2 flavours:

- Standard double precision interface.
- Single precision / null-terminated string interface.

Single precision is used, when much accuracy isn't needed (most situations). Null-terminated strings are used, when exact coordinates are needed:

- AirDistPos
- Coordinate2Location
- Coordinate2Node
- Location2Coordinate
- NodeCoordX
- NodeCoordY

An example:
Function FindNode uses double precision, while FindNodeA uses a mix of null-terminated strings and single precision parameters as required by Avenue.

The null-terminated string should use "." as decimal point no matter the actual windows setup. If the string doesn't hold a valid number, error -51 is returned.

The correct definitions are already entered in the sample avenue project and when referring to the

© 2013 RouteWare / Uffe Kousgaard

functions in Avenue code, the last "A" in the function name isn't needed.

Another function specifically for Avenue is ShowProgressAvenue 118. See the reference list for a description and the sample project on how to use it.

### 1.12.2 MapInfo / MapBasic

It is easy to use RW Net DLL with MapBasic. Just include the file called "rwnet_header.def" in your project. This is also shown in the sample applications.

An example of a full-blown MapInfo application based on RW Net DLL is "RouteFinder for MapInfo".

If you want to use TAB as output format, MITAB.DLL should be stored somewhere on the "windows path": C:\windows\system32\ or c:\program files\mapinfo\Professional\. The same folder as rwnet.dll will not do.

## 1.13 Advanced topics

### 1.13.1 Alpha parameter

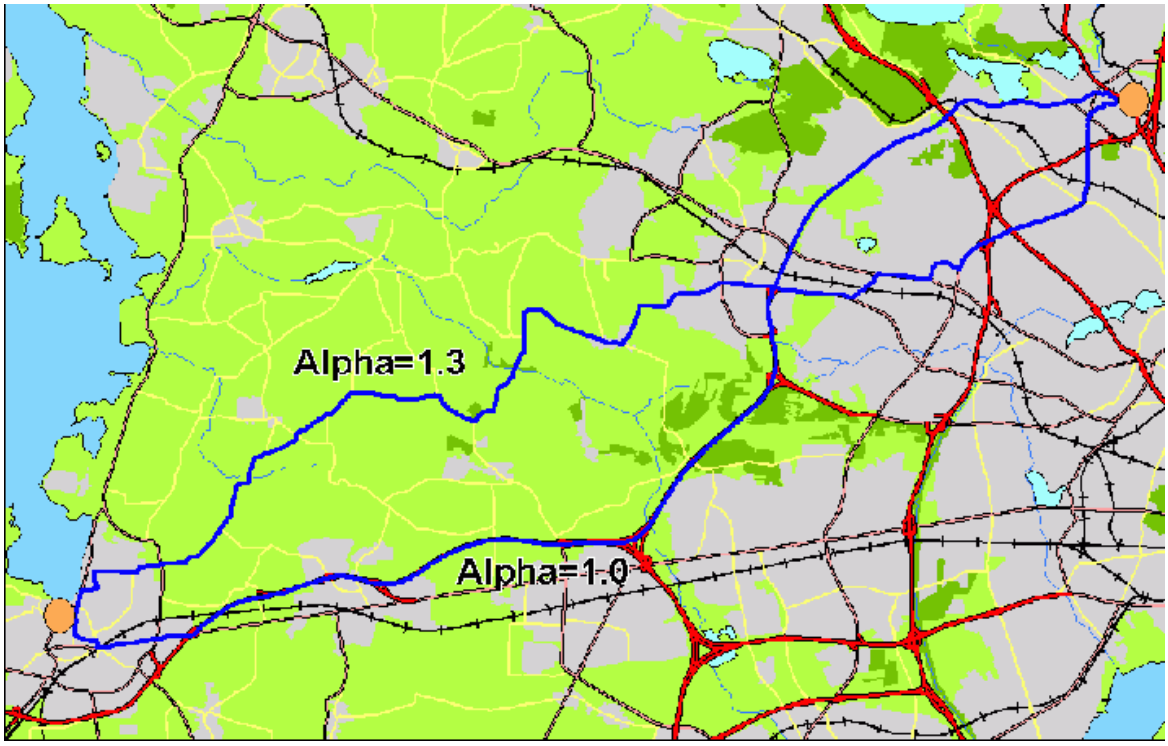
The alpha 57 parameter is used in function Route 106 / RouteDyn 106 (not for isochrones). It makes it possible to calculate routes faster than would normally be possible, and even faster if the user accepts only getting the correct result in <100% of the situations.

If you just want the result returned as fast as possible and still correct routes calculated, you can use function OptimumAlpha 98 to calculate the optimum value for Alpha.

By increasing the value of alpha to "1.3 x OptimumAlpha", you will get a route returned, which may be closer to a straight line between start and end node, but at the same time longer or slower than the true shortest or fastest route. Using 1.3 might improve calculation speed with a factor 5-10 depending on the network. See further down this page for more details on this.

On the map below an example is shown, where the exact shortest route is shown with alpha=1. On the same map, a route with alpha=1.3 is shown. The 2 routes a very different, but the actual length doesn't need to be that different. It is easy to see that the route with alpha=1.3 is closer to a straight line than the shortest route (which is also the fastest in this case).

© 2013 RouteWare / Uffe Kousgaard



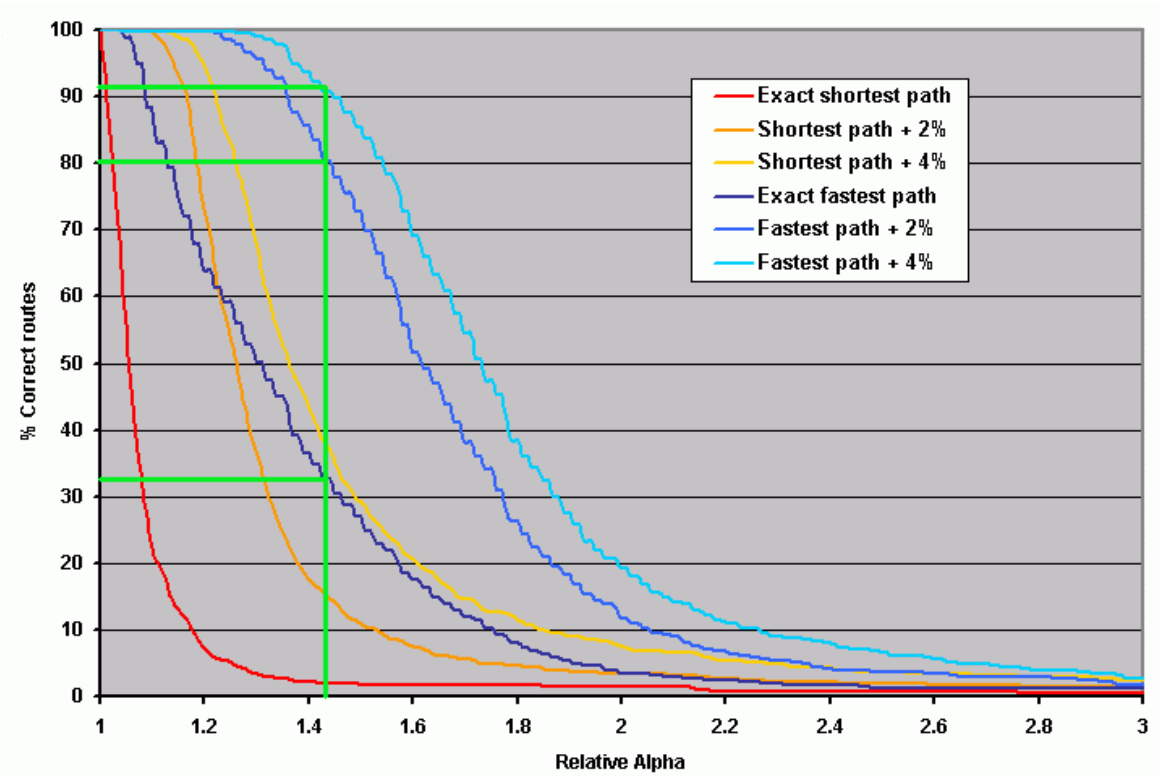

In the diagram below, the results from a test with TIGER data for Connecticut is presented. It shows how many % of the calculated routes are still correct, when alpha is increased relatively to the optimum value. The results are based on 1000 random node-to-node calculations.

Example: If the optimum value of alpha is 0.8 (for a fastest path calculation) and alpha=1.15 is used in the calculation the relative value of alpha is 1.15/0.8 = 1.44. From the diagram it can be seen, that 33% of the calculated routes will in fact still be correct, while 80-33=47% will be <2% longer, 91-80=11% will be 2-4% longer and the last 9% will be more than 4% longer than the exact fastest route. "Longer" of course refers to the time of the route, not the length.

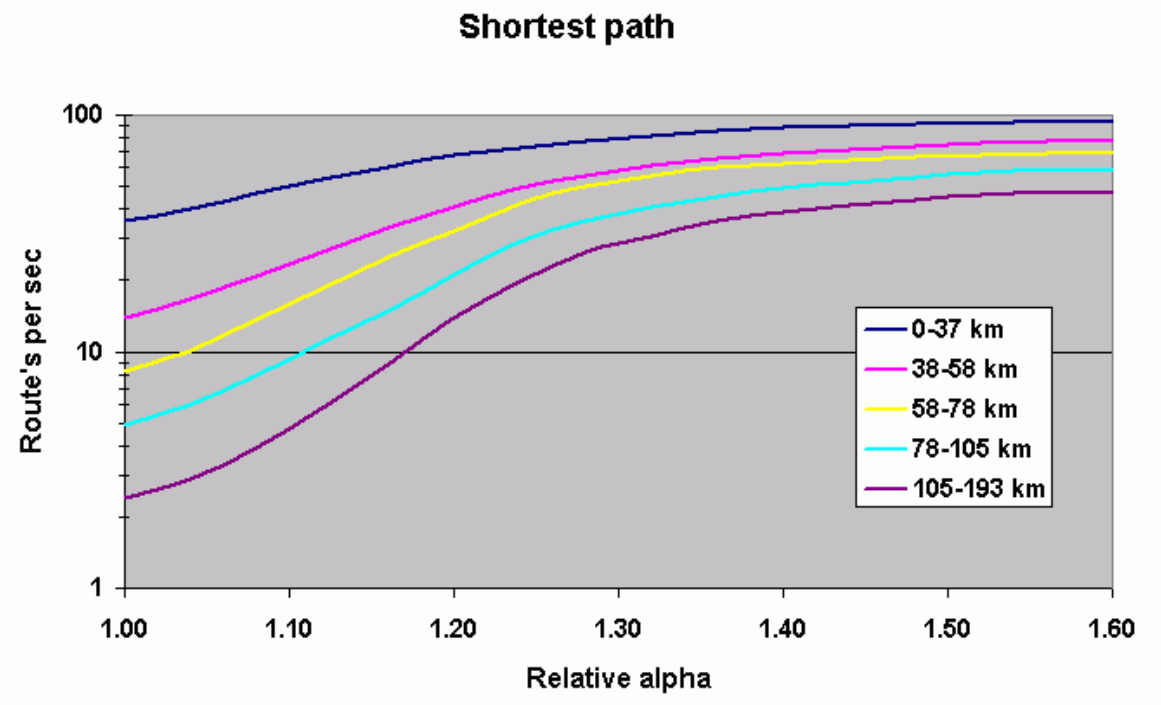
This diagram would look different, if you had another network, used another set of road link speeds, used another coordinate system or changed the setup in some other way. But the overall result would be the same.

This can also be used for adjusting the calculated length of a route once you know, how much the average route is longer than the correct result.

*© 2013 RouteWare / Uffe Kousgaard*

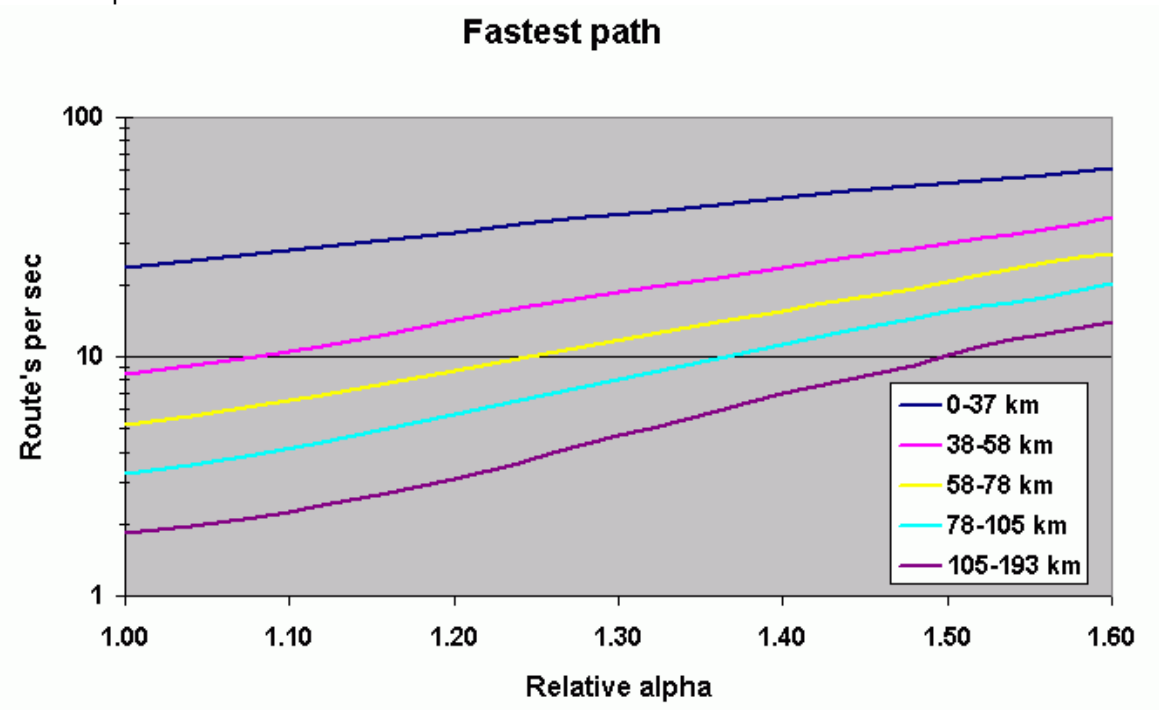



The actual impact on calculation time is shown in the diagram below. It can be seen that adjusting alpha has the biggest relative impact on long routes (105-193 km), where the number of route's calculated per second improves from 2.5 to 40 (16x faster) if alpha is increased from 1.0 to 1.4. Any further increase in alpha has only little impact. For short route's (0-37 km), speed increases from 35 to 90 route's per second.




© 2013 RouteWare / Uffe Kousgaard

The same diagram for the fastest path is shown below. The main difference from shortest path is the overall slower calculations (40% slower) and the smaller impact, when the relative value of alpha is increased. For long routes, the number of routes per second is only 4x higher, when using relative alpha=1.4 instead of 1.0. For short routes the difference is 2x.




### 1.13.2 Hardware recommendations

If you are working with large street networks and a performing multiple route calculations, it may be of interest to use the best possible hardware for the job. Testing has shown that it isn't easy to give exact recommendations, but this is our ordered list of importance:

1. Have enough RAM to avoid memory swapping.
2. Fast CPU speed.
3. Fast RAM access (high quality motherboard and system in general).
4. Intel Duron and AMD Athlon are slower than Intel Pentium / AMD64.
5. Multiple CPU's (requires multiple rwcalc objects to take advantage of them).

In order to have enough RAM, a quick rule of thumb is that each link in your street database requires 50 bytes of RAM. So with 1 million links, you need 50 MB RAM.

Doubling the speed of the CPU, doesn't make the calculations twice as fast, as the RAM will usually be a bottle-neck and the level-1 and level-2 caches of the CPU isn't large enough for most networks.

The functions IsoPoly2, IsoPoly3 and IsoPoly4 all requires a lot of RAM, when used with large street networks. Depending on the exact function parameters, this may be 3-4 time as much as the street network itself, i.e. another 150-200 bytes of RAM per link.

An example: 6.2 million links, IsoPoly2 function call with 3 drive time zones (2, 5 and 10 minutes) around 770 centers required 1.4 GB RAM (and took ½ hour). IsoPoly3 and IsoPoly4 are less demanding.

© 2013 RouteWare / Uffe Kousgaard

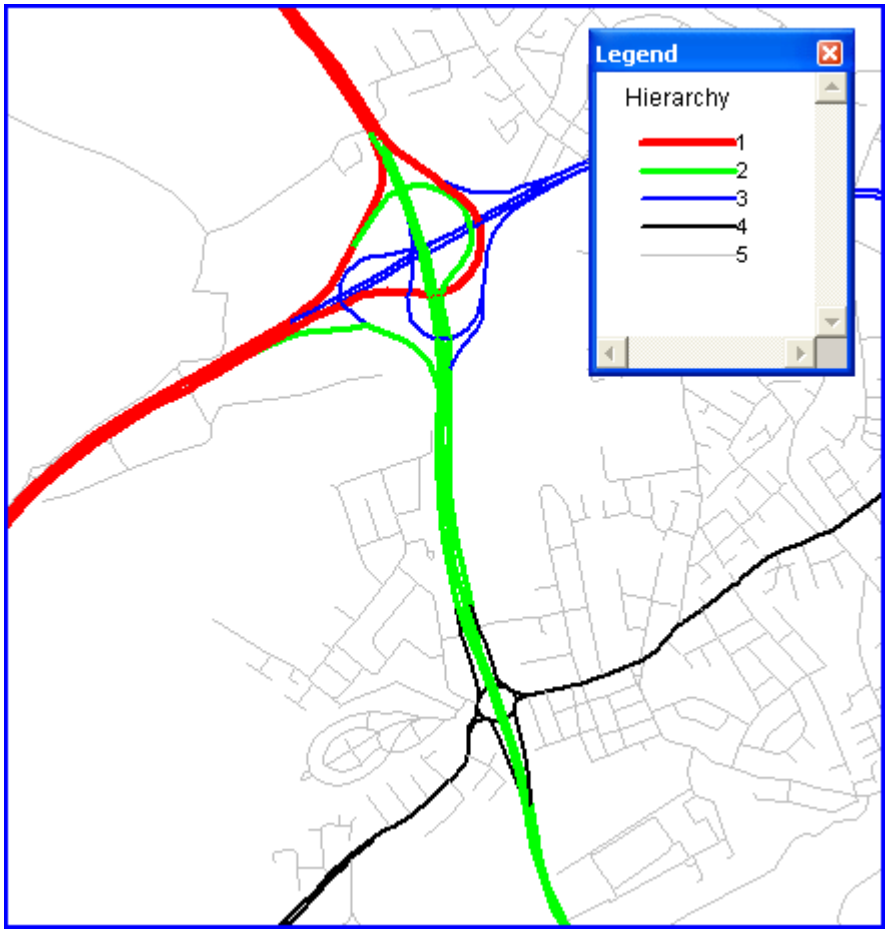
### 1.13.3 Hierarchical Routing

Some street databases has special attributes for the most important streets, the ones being used as part of long routes. This will typically be motorways, but can also be ferries, bridges and some minor streets which are required to have a connected network.

The advantages of restricting routes to these more important streets are:

- Much faster point-2-point route calculations for long routes (function Route 106, RouteDyn 106 and RouteList 111).
- Simpler routes, which doesn't make short-cuts via minor roads to make a long route a little shorter / faster.

The map below shows an example from TeleAtlas Multinet data with 5 layers of importance (hierarchies):




RW Net uses a method where the calculation of the route is restricted to level 1..X as soon as level X has been reached on the route unless you are within a certain distance Y of the final target. Then additional levels are included in the search again.

For the algorithm to work properly the parameter Y has to be supplied for levels 2 to 5. Level 1 (the toplevel) is of course always included in the search. The best values for these parameters depend on the geometric properties of the network and how the hierarchy attribute has been setup.

If you have less than 5 levels in your datasource - 3 for instance - use levels 1, 2 and 3.

© 2013 RouteWare / Uffe Kousgaard

If you choose small parameters values, a smaller part of the network is considered when you get close to the target and this improves calculation speed. The downside is you risk not finding the target at all (!), because there are no major streets within the limits you have defined. The solution to this problem is to re-calculate without the hierarchy setting or just use a more relaxed setting (bigger parameters values). Such re-calculations are costly and when choosing parameters it is important to find a balance between normal, fast calculations and the slow re-calculations.

In the documentation for HierarchyLevelSet 142 you can see suggestions for TeleAtlas and Navteq datasets and here is an example of how to use it:

```
rwcalc1.HierarchyLevelSet(145,90,40,7)
rwcalc1.hierarchy = true
dist = rwcalc1.Route(node1,node2)
if dist = -33 then
begin
  rwcalc1.hierarchy = false
  dist = rwcalc1.Route(node1,node2)
  rwcalc1.hierarchy = true
end
```

The logic above is already part of the RouteList 111 function internally.

See also functions HierarchyCreate2 142, HierarchyLoad 143, HierarchyLevelSet 142 and Hierarchy 141.

### 1.13.4 Isochrones (polygon)

A typical use of RW Net is to create isochrones around one or more center facilities. This can be based on either time or distance (or more generally cost). When such an isochrone is calculated, the cost is only defined on the street network itself. But typically there is a wish to have the isochrone presented as a polygon layer, often with thematic colours.

Many approaches can be used to transform the link-based results into polygons, but it is not possible to create a transformation, which will be acceptable in every single scenario - simply because it isn't clear what the cost is once you leave the road and what street segment you want to start from.

In RW Net the "off-road" cost is 0, so that "*the cost inside a drivetime polygon somehow describes the cost of driving from the nearest road to the target facility*". The nearest road does, however, not always has to be the one with the lowest cost !

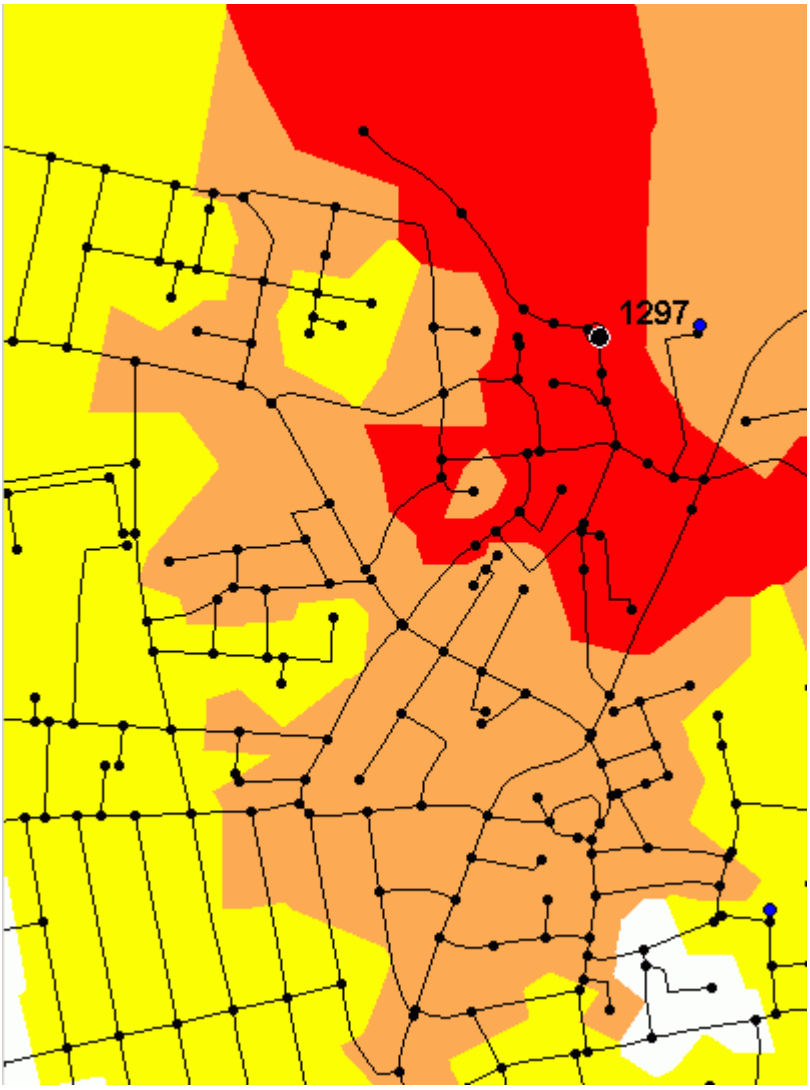
In RW Net an approach is chosen which uses voronoi polygons created around the nodes in the street network. This is a very fast method, but has some drawbacks in the case of long street sections without any breaks / nodes. To make the result more correct, it is possible to dynamically add additional nodes along long street sections (AddNodes 57).

Calculation of the voronoi polygons uses 2D formulas for the trigonometric calculations, so isochrones based on spheric coordinates will not be 100% correct, but it is unlikely that this will ever be a problem for normal use.

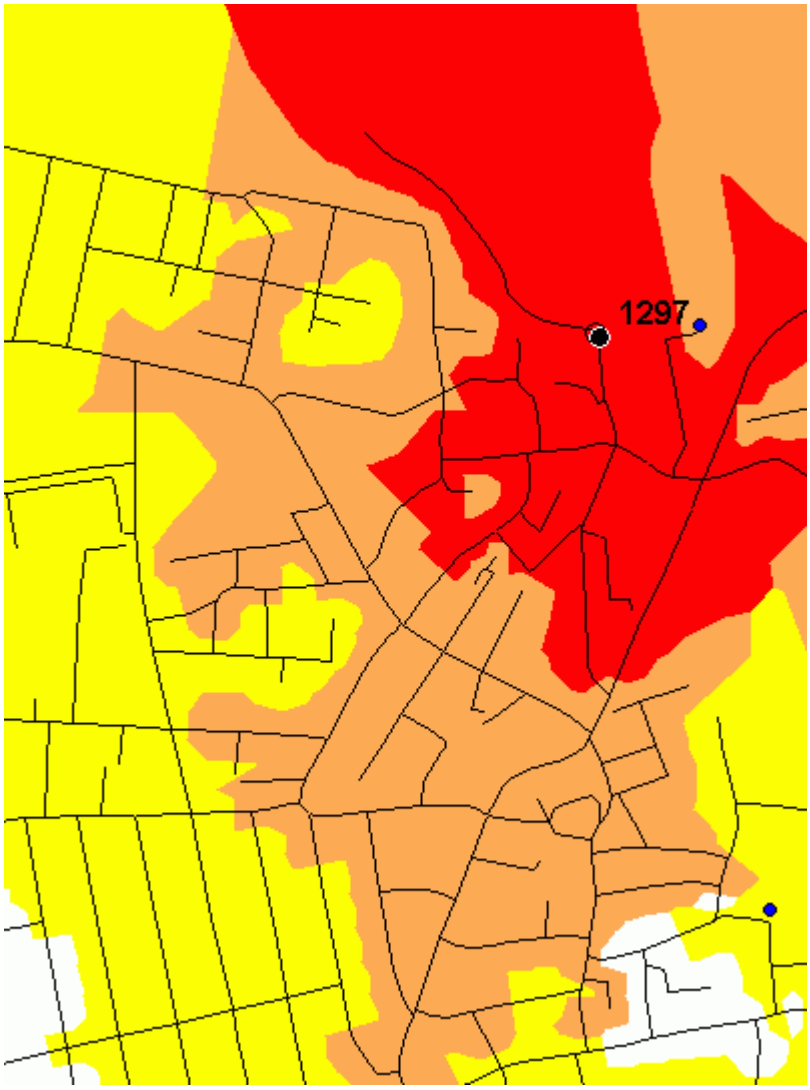
See also function IsoPoly2 81, IsoPoly3 83, IsoPoly4 84 and property AddNodes 57.

Below you can see 2 maps. The first shows an isopoly2 calculation with isochrones at 1, 2 and 3 km. Only the true nodes of the network has been used in the calculation (shown on the map). On the second map, additional nodes for every 50 meters has been added and a more exact and smooth polygon is the result.

© 2013 RouteWare / Uffe Kousgaard




© 2013 RouteWare / Uffe Kousgaard




### 1.13.5 Progress events

Several progress events are available. When using the OCX / VCL / assembly version, you are responsible for showing the actual progress on screen, while the DLL version has the dialogs built-in (see ShowProgress 118 and ShowProgressAvenue 118).

All events step from 0 to 100 and back to 0. Every step (0, 1, 2, 3....) is generally triggered 10 times to allow smooth processing of windows messages and the events step as smooth as possible.

These are the functions with events attached:

RWnetBase object:
AttributeCreate2 58, AttributeLoad 59, NodeCreate 91*, NWcreate 94*, NW3Dnodes 93*, NWexport 146, NWload 97, Overpasses 147* and RoadnameCreate 103.

RWcalc object:
CPP 132*, FindRoundAbout 140, Join 143, Join2 144, Route 106, RouteDyn 106, TSP2 120* and TSP2dyn 122*.

© 2013 RouteWare / Uffe Kousgaard

RWcalc object, OnIsoChroneProgress event:
IsoCost 75, IsoCostDyn 75, IsoCostDynLocationList 76, IsoCostDynLocationListN 77, IsoCostMulti 77, IsoCostNodeList 78, IsoCostNodeListN 78, IsoLink2 80, IsoLink2Dyn 80, IsoPoly2 81, IsoPoly3 83, IsoPoly4 84.

Functions marked with a * can be interrupted by changing the StopProcess parameter or clicking cancel (DLL).

### 1.13.6 Shortest / Fastest path

In the sample code 14 we saw how the fastest route was defined by setting:

```
RWcalc1.CostDist = 0
RWcalc1.CostTime = 1
```

This means the weight on distance is 0 and the weight on time is 1. If we want the shortest path, we just swap 1 and 0.

But it is also possible to use a linear combinations of these values and still get access to both the time and length of a calculated route:

```
RWcalc1.CostDist = 0.4
RWcalc1.CostTime = 1
RWcalc1.ExtraDist = 0
RWcalc1.ExtraTime = 1
```

Now the route is optimized according to minimizing "1.0 * time + 0.4 * distance". This means most weight is on time, while the length of the trip is also taken into consideration, so that very long routes are not chosen, just because they are slightly faster.

In order to get the time and length of the route, use this method:

```
Cost = RWcalc1.Route(node1,node2)
RouteTime = RWcalc1.GetNodeExtra(node2)
RouteDistance = (Cost-RouteTime) / 0.4
```

### 1.13.7 Turn Restrictions

Calculations can be performed calculations in two modes: With or without support for turn restrictions. Before a network is loaded, you will have to decide which mode to use by setting property TurnMode 124. The internal working of the two modes are quite different, which means performance and RAM requirements are also different:

- Turn restriction mode is app. 2 times as slow as normal mode.
- Allowing U-turns 126 slow the calculations down further (app. 30%).
- Turn restriction mode takes up more RAM - see Function NWload 97 for details.

When adding restrictions to the network you can choose between either a 100% restriction (a negative value) or a delay expressed in the same unit as costs. If cost is measured in minutes, using "1" would add a delay of 1 minute for a specific turn. This could be e.g. a left-turn. Don't use turn-delays combined with shortest path routes - it just wouldn't make sense to add a 1 km to the route, since the reported distances would be wrong.

When querying a route the delay is added just after the turn has been performed: A delay of 2 minutes at node B has no impact on the cost of going from node A to node B, whereas if you move on to the next node, the 2 minutes are reflected in the next node on the route.

© 2013 RouteWare / Uffe Kousgaard

### Adding turn restrictions

Restrictions can be added, updated and deleted using the same function (TurnRestriction 125). It is also possible to add restrictions to a normal 4-road intersection in one step by using function TurnStandard 126. Then no turns are allowed except driving straight through.

It is possible to import/export turn restrictions for easier retrieval at a later time. Use function TurnImport2_TXT 123 and TurnExport2_TXT 123 for this. Similar functions exist for binary formats. All currently defined turn restrictions can be removed with procedure TurnReset 125.

With the release of RW Net 2.33 a new set of functions has been added to allow mandatory turns and use of external ID's in defining the turn restrictions. Support for complex turn restrictions has also been prepared.

### Querying a route

In normal mode, each intersection/node can only be visited once on a route. In turn restriction mode each link can only be travelled once in each direction, but the intersections can be visited several times. This has some implications when mixing the two modes in the same application:

If you call Route with the same node1 and node2, you will get an error code in normal mode, while turn restriction mode will return a valid path, which may be just one link travelled in both directions (if you allow U-turns) or something a little more complicated otherwise depending on your network.

In normal mode you would use a loop like this to query the cost along a route (pascal-code):

```
route(node1,node2);
length:= RouteFind(node2);
for i:= length downto 1 do
begin
  writeln('Node: ',RouteGetNode(i),' ',GetNodeCost(RouteGetNode(i)),'
km');
end;
Writeln('Total cost: ',GetNodeCost(node2));
```

Since a node can be part of a route more than once in turn restriction mode, you can not use the loop above, but has to use a loop like this instead. The advantage is that this loop will also work in normal mode and produce the same results as above:

```
Route(node1,node2);
length:= RouteFind(node2);
for i:= length downto 1 do
begin
  write('Node: ',RouteGetNode(i));
  if i=length then writeln(' 0 km') else
    writeln(' ',GetLinkCost(RouteGetLink(i)),' km');
end;
Writeln('Total cost: ',GetNodeCost(node2));
```

## 1.13.8 Very large networks

In some situations you may want to work with very large street networks, which for one reason or the other is too large to handle (lack of RAM, you want faster calculations, licensing issues etc.).

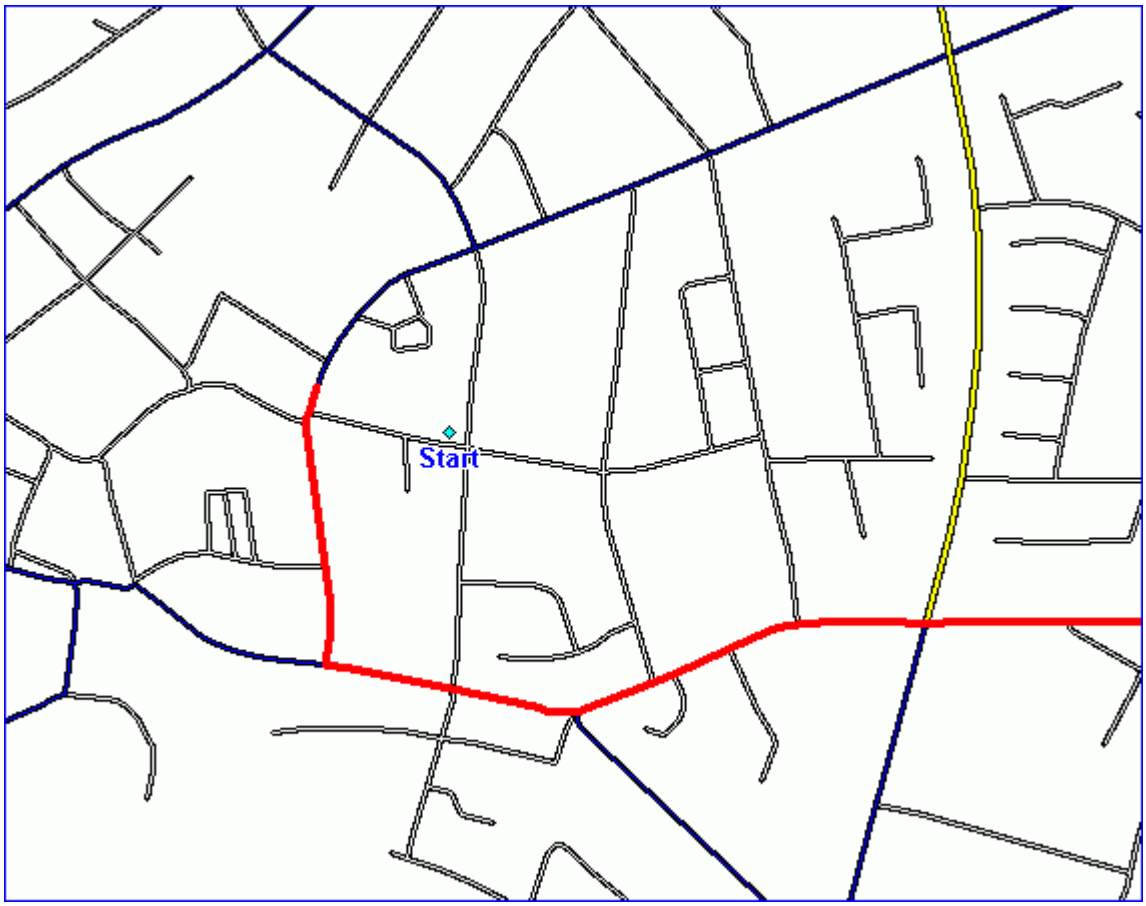
One solution to such a problem is to delete the less important streets from the street database. This can typically decrease the size of a database significantly and most commercially available databases already has attributes for this kind of operation (e.g. select * where street_level>2). This also normally ensures that the rest of the streets still constitute a connected street network without

© 2013 RouteWare / Uffe Kousgaard

any subnets.

The drawback is of course that all routes are limited to the selected street and driving directions very close to the start and end of the route will be missing. It will however still be possible to show a map with all streets and a signature of some kind, which shows the target, which is enough for many purposes.

The picture below shows an example, where the white streets has been omitted from the actual street network in the routing calculations, but they are still shown on the map, so the route starts a little away from the real start point:




### 1.13.9 Limits

These functions makes it possible to calculate routes, while taking certain limitations for the links into account. 2 types are available:

1. A scalar quantity such as a maximum weight, height, width etc. If the limit for a certain link in the network is 100 and you calculate a route for a vehicle with a value >100, that link will be avoided in the route. It is mandatory to scale your limits into the 1-255 interval.

2. A bit pattern for defining special links such as ferries, toll roads etc. which you may want to avoid in your routing. If the limit for a link is 3 = 00000011 it may mean it is both a ferry and a toll "road" (most ferries are not free, so that seems logical). If your value has either bit 1 or 2 set, that link will be avoided in the route. It is possible to define 8 such bits within each limit.

© 2013 RouteWare / Uffe Kousgaard

For both types, a link value of 0 means no limitations at all.

A maximum of 9 such limitations can be created in files and a maximum of 9 of these can be loaded into memory at a time.

Use function LimitCreate 84 to setup the files and LimitLoad 86 (type 1) / LimitLoad_bitpattern 86 (type 2) to load the limit files into RAM.

Use SetLimit 116 to define a value for the route about to be calculated.

Example with maximum height expressed as decimeters:

```
RWnetBase1.NWLoad
rwnetbase1.LimitLoad(1,1)   // we load it into position 1 from
"limit1.bin"

rwcalc1.SetLimit(1,0)   // no limit (default)
cost1 = rwcalc1.Route(node1,node2)

rwcalc1.SetLimit(1,40)  // 40 dm = 4.0 meter
cost2 = rwcalc1.Route(node1,node2)

rwcalc1.SetLimit(1,42)  // 42 dm = 4.2 meter
cost3 = rwcalc1.Route(node1,node2)
```

With the proper sample data, this will result in 3 different routes. The first route is the shortest, the second is longer and the last one is the longest (most restrictive limitation).

### 1.13.10 Performance

Performance of the core routing functionality is important to anyone considering using large street networks.

From the TIGER data (USA) we have extracted a 4,500,000 links street network covering the central part of USA (1600 x 1600 km).

The test was done with this setup:

- Dynamic segmentation, 2 calls to function Coordinate2location 62
- Generation of driving directions, function RouteList 111
- Latitude/Longitude coordinates
- Alpha = 1.3
- Fastest route
- Caching of all data, except file coord3.bin
- Computer was an AMD Athlon 64 3200+ with 2 GB RAM.
- The network required 450 MB harddisk space.

We have randomly calculated a large number of routes.

Average time per route:

| | Short routes | Long routes | RAM usage |
|---|---|---|---|
| Normal mode | 422 msec | **1358 msec** | 268 MB |
| Turn restriction mode | 645 msec | 2876 msec | 313 MB |

© 2013 RouteWare / Uffe Kousgaard

Long routes can be as long as 3100 km, but most will be a lot shorter.
Short routes are within the same network, but start and end location is restricted to a 500 x 500 km rectangle with 450,000 links.

If you use a dataset with attributes for defining a hierarchy 36, calculation time can be much faster (just 100-200 ms). This is available in RW Net Pro.

**Multi-threading**

The normal-mode, long route test from above has been executed on a number of different computers and with different number of threads. As can be seen from the table below, true dual CPU gives more processing power than a hyper-threading enabled (HT) CPU. Of course this requires simoultaneous routing requests and an application allowing for it, such as RW NetServer:

Number of routes per minute:

| CPU | 1 thread | 2 threads | 3 threads | 4 threads |
|---|---|---|---|---|
| Dual P3-866 | 19 | 32 | | |
| P4-3000 (HT) | 34 | 46 | | |
| AMD Athlon 64 3200+ | **44** | 44 | | |
| Dual Xeon 3.2 GHz (HT) | 40 | 66 | 82 | 92 |

## 1.14 License terms

A license gives access to all available versions: Delphi 5-8, 2005-2010 VCL, Active-X, .NET 1.1 and 2.0 assembly and 32-bit DLL. Within the first year licensor has full access to updates, new versions and e-mail support. After that period licensor can at any time subscribe for support and updates for one more year.

Support is rendered on an e-mail basis and includes answering questions, which do not involve writing source code of >10 lines. Only persons with a license can receive e-mail support. Support is included for the first year, but can be extended for one more year at a time.

Licensor gets free updates for RW Net as long as licensor is entitled to support.

Licensor is allowed to:

- Use RW Net for as long as he/she likes and as long as he/she doesn't violate this license.
- Distribute applications created with RW Net on a royalty free basis.
- Use RW Net in desktop applications and systems running for a specific audience on a LAN / intranet.

Licensor is not allowed to:

- Distribute applications outside it's own organization, which competes directly with RouteWare's own applications: RouteFinder, RW NetServer and ToolBox.
- Wrap up RW Net in component-like structures and distribute it.
- Use RW Net in applications and systems available over a WAN / internet (A).
- Use functions NWcreateMIF/SHP/TAB/CGF in applications distributed to users without a license for RW Net (B).

For an additional fee, it is possible to avoid limitations (A) and/or (B). Refer to RouteWare website for more info.

If licensor holds a personal license, he/she can either:

© 2013 RouteWare / Uffe Kousgaard

1) Have only 1 specific person using RW Net on as many computers as he/she like or
2) Install it on 1 specific computer and let several persons use it from there, but not at the same time (support is still only given to 1 person)

If licensor holds a site license, it allows an unlimited number of persons at licensors site to use RW Net at the same time. Ask RouteWare for enterprise-wide licenses.

Users of the 32-bit DLL, .NET and Active-X version of RW Net are issued a personal password to activate the software. This password must not be readable to end-users of distributed applications. It is the responsibility of licensor to ensure that this is taken care of.

**The usual legal stuff**

All copyrights belong to RouteWare (Uffe Kousgaard).
Disassemble or reverse engineering of RW Net binaries are not allowed.
Licensor is not allowed to install RW Net on a network drive or shared drive except for backup purposes.
Licensor is not allowed to sell or in any other way hand over the right to use the software to any other party.
RouteWare is not responsible for any problems, direct or indirect, which RW Net may cause - no matter what the reason may be.
Any problem / error will be corrected as fast as possible within normal business hours. If RouteWare is not able to correct problems, which to a severe degree affect the functionality of the software, a refund is made, which matches the degree to which the software doesn't function properly. This refund is based on what the licensor has paid within the last 12 months and cannot exceed this amount.

Updated Oct 2004

*© 2013 RouteWare / Uffe Kousgaard*

# Part II

# Error codes

# 2 Error codes

All functions return a value, which can either be the result from the function or an error code:

| | |
|---|---|
| 0 | No error |
| -10 | Network not loaded |
| -11 | RW Net trial license has expired |
| -12 | Network too big for license or input file empty |
| -13 | Invalid shapefile |
| -14 | Too many links at one node (no more than 1900) |
| -15 | Wrong version code in network file |
| -16 | Function not available in turnmode |
| -17 | CGF file is password protected |
| -18 | Function only available in turnmode |
| -19 | External ID not open - see externIDopen 70 |
| -20 | Field name not found in file |
| -21 | Field isn't of type number/decimal 11.3 |
| -22 | Field is not numerical |
| -30 | Number out of bounds |
| -31 | Couldn't load network: wrong path, file not found etc |
| -32 | File couldn't be found, opened, created etc. |
| -33 | Node2 / link2 not found |
| -34 | No node index created |
| -35 | Route not ready |
| -36 | Route index out of bounds |
| -37 | No results calculated |
| -38 | Nodes in subnet or node=0 |
| -39 | Function ExtraVarCreate 71 hasn't been called |
| -40 | General error |
| -41 | Out of memory |
| -42 | Internal error |
| -43 | Function not available in Free/Standard version (OCX only) |
| -44 | No spatial index found (spatialindex.bin) |
| -45 | No link found |
| -46 | No coordinate file found (coord3.bin) |
| -47 | Not possible with encrypted files |
| -48 | Road file is already open |
| -49 | Road file is not open |
| -50 | Cancelled by user |
| -51 | Number of records in road file doesn't match network |
| -52 | The network contains subnets |
| -53 | (Not implemented yet) |
| -54 | No solution found, increase factor in function CPP |
| -55 | One or more nodes or locations are not defined in list |
| -56 | Only shortest or fastest route can be processed with this function |
| -57 | Function VRPcreate has not been called |
| -58 | VRP solution not possible - limits too tight |
| -59 | Isochrone steps too big - no output |

© 2013 RouteWare / Uffe Kousgaard

| | |
|---|---|
| -60 | Limit files has wrong size |
| -61 | Invalid "lambda" parameter |
| -62 | Coordinate not within valid window |
| -63 | Hierarchy need to be loaded |
| -64 | Call SetLinkResult 116 first |

See also function ErrorMsg 68.

© 2013 RouteWare / Uffe Kousgaard

# Part III

# Check list

# 3 Check list

Sometimes a calculation returns a different result (route) compared to what you expected or you get no route at all (error code -33 / -38). Both situations are due to issues somewhere in the network and these can be hard to locate.

This is a list of things to check:

- Look inside network_report.txt (generated when calling NWcreate):

1. Are you using the latest version of RW Net?
2. Is the coordinate unit correct?
3. Should Z-level information have been applied?
4. Do some of the objects have errors?
5. Is the average object length reasonable?

- Check for network errors. See also data sources 12.

- Do you create and load your network in the correct order? See flowchart 26.

- Do you check the return code from all function calls?

- Reduce the setup as much as possible:

1. Call Attributeload(false) (removes oneway restrictions) or
2. Skip calling Attributeload (removes oneway restrictions and mode restrictions)
3. Set Turnmode = false
4. Set Hierarchy = false
5. Skip Limits
6. Use SetShortest instead of SetFastest
7. Set Alpha = 0

If this solved the problem, enable these again one by one, until it fails. Then you know where to look.

- Create maps in your GIS identifying the problem:

1. Look at basic map for no physical connection (missing bridge / ferry etc)
2. Create a thematic map of one-way directions and closed links
3. Create a thematic map of attribute field
4. Create a thematic map of hierarchy attributes etc.
5. Call TurnExport2_GIS 122 to view turn restrictions
6. Call FindCloseNodes 139 from RW Net Pro
7. Call NodeLinkCheck 146 from RW Net Pro.
8. Call SubNet 150 from RW Net Pro
9. If you don't have RW Net Pro version, try Isolink2 80 with a large stepsadd() value

If you have multiple points (TSP functions or RouteList with >2 points), it can be tricky to locate which point makes the trouble. One way is to calculate with 2 points first. If that works OK, try with 3 points, then 4 points etc, until the problem pops up. Now the problem is usually somewhere near the last point being added.

© 2013 RouteWare / Uffe Kousgaard

# Part IV

# Reference

# 4 Reference

## 4.1 AddNodes

### Property AddNodes: double;

Setting this property to a value >0 affects the output of function isopoly2 81, isopoly3 83 and isopoly4 84.

If the value of AddNodes is, say 1 km, and a link is 3.6 km long, additional nodes will be inserted at 0.9, 1.8 and 2.7 km in the generation of isochrones. No additional nodes are added if the link is shorter than 1 km.

Watch out for using very low values or the number of additional nodes gets so big, that the generation of isochrones runs out of memory.

See also separate discussion 37 on isochrone calculation.

Use same length unit as defined in property units 126.

*ActiveX / VCL / CLX component:* RWcalc

## 4.2 AirDistNode

### Function AirDistNode(node1,node2: longint): double;

Calculates the as-the-crow-flies distance between two nodes. Uses the current coordinate system.

*Possible error codes:* -10 -30
*ActiveX / VCL / CLX component:* RWnetBase

## 4.3 AirDistPos

### Function AirDistPos(Xlong1,Ylat1,Xlong2,Ylat2: double): double;

Calculates the as-the-crow-flies distance between two geographic positions. Uses the current coordinate system.

*Possible error codes:* -30
*ActiveX / VCL / CLX component:* RWnetBase

## 4.4 Alpha

### Property Alpha: double;

Alpha is used in function Route and 0<=alpha<=4.

See seperate description of alpha 32.

In the freeware version alpha is always 0.

© 2013 RouteWare / Uffe Kousgaard

*ActiveX / VCL / CLX component:* RWcalc

## 4.5 AttributeCreate

### Function AttributeCreate(filename: string): integer;

This function performs the same operation as AttributeCreate2 58, but creates the attribute file from a text file instead. This is an alternative function, if your data are not in a DBF/TAB format.

The text file shouldn't contain any other values but the attribute, that is one positive integer on each line in the file.

Filename should be without path information, since that is obtained from property Directory 67.

*Possible error codes:* -32 -40
*ActiveX / VCL / CLX component:* RWnetBase

## 4.6 AttributeCreate2

### Function AttributeCreate2(filename, fieldname: string; fieldindex: integer): integer;

From the specified filename a file called attribute.bin is created, which holds all attributes regarding the links. See definition of attributes here 27.

The file should be of type DBF (version 3) or DAT file (part of a TAB file).

In the case of a DAT file, the corresponding TAB file needs to be present.
If you supply fieldindex=0, the fieldname will be used. If you supply a fieldindex (1..) it will be used for reading from the file.
If the field, you are pointing at, is a floating point field, it will be rounded.
If any value is negative or >65535, it will be set to 0.

Filename should include a fully qualified path.

Attribute.bin simply holds the values from the input file and so it is possible to write directly to this file if needed. It consists of 16-bit unsigned integers. You may even create the file from scratch yourself, if that suits you better, but you miss being able to encrypt the file.

Use OnAttributeCreateProgress event for tracking progress.

**Examples:**
AttributeCreate2("roads.dbf","",1)
> Use first field from *roads.dbf* file.

AttributeCreate2("roads.dat","attrib",0)
> Use field *attrib* from *roads.dat* file. File *roads.tab* needs to be present.

See also AttributeCreate 58

© 2013 RouteWare / Uffe Kousgaard

**Alternative method:**

If you prefer not to use the road classes, because you already have the exact time or speed on each link in your network, you can use the functions ReadSpeed 100, SetLinkSpeed 116 or SetLinkTime 117 instead. For one-way information, you can use the function CloseLink 60 to change one-way status. For entering roundabout status, there is no other way but the attribute file.

*Possible error codes:* -20 -22 -30 -32 -40
*ActiveX / VCL / CLX component:* RWnetBase

## 4.7 AttributeLoad

### Function AttributeLoad(oneway: boolean): integer;

Calculates the time on all links according to attribute.bin (roadclass, mode), speed and Units. Should be called whenever one of these have been changed.

If oneway is false, all links will be two-way streets (for instance for pedestrians). For other modes specify true.

Use OnAttributeLoadProgress event for tracking progress.

*Possible error codes:* -10 -31 -40
*ActiveX / VCL / CLX component:* RWnetBase

## 4.8 AttributeLoad2

### Function AttributeLoad2(oneway: boolean: filename: pchar): integer;

Same function as AttributeLoad 59, except you can also include the fully qualified filename, if you have additional attribute files.

Example: attributeload2(true,"c:\app\attributefiles\attrib_cars.bin")

*Possible error codes:* -10 -31 -40
*ActiveX / VCL / CLX component:* RWnetBase

## 4.9 AttributeSet

### Function AttributeSet(link: integer; newattrib: smallint): integer;

Changes the attribute for a single link. The change only takes effect on disk. You should still call AttributeLoad afterwards. AttributeCreate should already have been executed.

If EncryptionKey 138<>0, you will get error -47 in return.

*Possible error codes:* -32 -40 -43 -47
*Versions:* Standard Pro

© 2013 RouteWare / Uffe Kousgaard

*ActiveX / VCL / CLX component:* RWnetBase

## 4.10 BestNode

### Function BestNode: integer;

See function Route for a description.

*Possible error codes:* -10 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.11 CheckLink

### Function CheckLink(link: integer): boolean;

Returns true if the link number is valid.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.12 CheckNode

### Function CheckNode(node: integer): boolean;

Returns true if the node number is valid.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.13 CloseLink

### Function CloseLink(link: integer; code: smallint): integer;

Changes the status of the specified link for network loaded in RAM:

0: Two-way street
512: One-way street, which may only be travelled in the digitised direction.
1024: One-way street, which may only be travelled in the reverse of the digitised direction.
1536: Closed street.

This overwrites information loaded with AttributeLoad and after a call to AttributeLoad it is also overwritten.

See also GetOpenStatus 74 and CloseLinkFixed 61

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.14 CloseLinkFixed

### Function CloseLinkFixed(link: integer; code: smallint): integer;

Same as CloseLink 60, except it changes the status in the default attribute.bin file on disk. Status of network in RAM is not changed.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.15 Coord

### Property Coord;

Coord defines the coordinate system used when creating the network and querying road distances etc. The following are defined:

XYMiles, XYKm, XYInch, XYft, XYyd, XYmm, XYcm, XYm, XYsurveyft, XYnmi, LatLong, XYlink, XYchain, XYrod, XYdm, XYpoint

LatLong is default.

You should set this property before calling function NWcreate 94 and function NWload 97.

See also Units 126 and separate discussion 28.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.16 Coord3

### Property Coord3: boolean;

(From version 2.17, coord3 is always true internally, so setting it false has no effect.)

If coord3 is true, a file called coord3.bin is created at the same time as the network is created - NWcreate 94

See separate discussion on file structure 29.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.17 Coord3Cache

### Property Coord3Cache: boolean

If true, the content of coord3.bin is cached in memory, when function NWload is called. Otherwise it is read from disk when needed. Changing the value after the network is loaded has no effect.

This generally effects these functions:

© 2013 RouteWare / Uffe Kousgaard

- IsoLink2
- IsoLink4
- IsoPoly2, if addnodes<>0
- IsoPoly4, if addnodes<>0
- RouteFile*
- RouteList

Depending on the speed of your harddisk, it can have a positive effect to cache the file in memory, but it can also be a very big file to cache.

Default value is false.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.18 Coordinate2Location

### Function Coordinate2Location(const XLongV,YLatV: double; var link: integer; var percent: double; var side: integer; var dist,XLongNew,YLatNew: double): integer;

Finds the nearest location based on a set of coordinates. A location is a position along a link. This is described as a percentage (0 <= percent <= 1) along the link ID.

The side of the link is also returned in relation to direction of digitization: Right (0) or Left (1).

The air distance and coordinates of the location is returned in dist and XLongNew & YLatNew.

You will get error -30 if you supply a set of lat/long values not within valid range, i.e. -90..90 and -180..180.

There is quite some calculation work involved in this function, so you should consider to cache the results (store the location in an array/database), if the same coordinates are otherwise supplied over and over. This can have a significant impact on total calculation time.

Typical errors when using this function is using (0,0) as input or swapping coordinates. Both will slow down the calculations and return wrong results.

See also Location2Coordinate 87, Coordinate2LocationSimple 63, CoordinateWindow 66 and SpatialIndexLinksPerCell 119.

*Possible error codes:* -10 -30 -40 -43 -44 -46 -62
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.19 Coordinate2LocationSimple

### Function Coordinate2LocationSimple(const XLongV,YLatV: double; var link: integer; var percent: double): integer;

Same as Coordinate2Location 62, just with fewer parameters returned. Percentage is also returned as 0 < percent < 1.

*Possible error codes:* -10 -30 -40 -43 -44 -46 -62
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.20 Coordinate2LocationIgnoreGet

### Function Coordinate2LocationIgnoreGet(const link: integer): boolean;

Returns if link is being ignored (true) or not (false) in function Coordinate2Location.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.21 Coordinate2LocationIgnoreNodes

### Procedure Coordinate2LocationIgnoreNodes(mode: integer);

After several links has been defined for being ignored in calls to function Coordinate2Location, you can call this function and then all nodes, which is only connected to ignored links, will also be ignored in function Coordinate2Node (mode=1).

If mode=2, all nodes will be ignored if they are connected to any ignored link.

This can for instance be used, if you want to ignore all motorway sections in both Coordinate2Location and Coordinate2Node functions. Then you first define all these with Coordinate2LocationIgnoreSet() and then call this function.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.22 Coordinate2LocationIgnoreReset

### Procedure Coordinate2LocationIgnoreReset(const link: integer);

Defines that Link should no longer be ignored, when calling function Coordinate2Location.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.23 Coordinate2LocationIgnoreResetAll

### Procedure Coordinate2LocationIgnoreResetAll;

Defines that no links should be ignored, when calling function Coordinate2Location. This is also the status after a call to function NWload.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.24 Coordinate2LocationIgnoreSet

### Procedure Coordinate2LocationIgnoreSet(const link: integer);

Defines that Link should be ignored, when calling function Coordinate2Location 62.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.25 Coordinate2LocationIgnoreSetClosedLinks

### Procedure Coordinate2LocationIgnoreSetClosedLink;

Defines that all closed links (GetOpenStatus 74 = 1536) should be ignored, when calling function Coordinate2Location 62.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.26 Coordinate2Node

### Function Coordinate2Node(const XLongV,YLatV: double; var node: integer; var dist: double): integer;

Finds the nearest node based on a set of coordinates. Returns node number and air distance in node and dist. This function replaces FindNode and FindNodeFast.

You will get error -30 if you supply a set of lat/long values not within valid range, i.e. -180..180 and -90..90.

There is quite some calculation work involved in this function, so you should consider to cache the results (store the node numbers in an array/database), if the same coordinates are otherwise supplied over and over. This can have a significant impact on total calculation time.

Typical errors when using this function is using (0,0) as input or swapping coordinates. Both will slow down the calculations and return wrong results.

See also CoordinateWindow 66 and SpatialIndexLinksPerCell 119.

*Possible error codes:* -10 -30 -40 -44 -62
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.27 Coordinate2NodeIgnoreGet

### Function Coordinate2NodeIgnoreGet(const node: integer): boolean;

Returns if Node is being ignored (true) or not (false) in function Coordinate2Node

*ActiveX / VCL / CLX component:* RWnetBase

## 4.28 Coordinate2NodeIgnoreReset

### Procedure Coordinate2NodeIgnoreReset(const node: integer);

Defines that Node should no longer be ignored, when calling function Coordinate2Node.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.29 Coordinate2NodeIgnoreResetAll

### Procedure Coordinate2NodeIgnoreResetAll;

Defines that no nodes should be ignored, when calling function Coordinate2Node. This is also the status after a call to function NWload.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.30 Coordinate2NodeIgnoreSet

### Procedure Coordinate2NodeIgnoreSet(const node: integer);

Defines that Node should be ignored, when calling function Coordinate2Node.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.31 CoordinateExtract

### Function CoordinateExtract(const link,index: longint; var x,y: double): longint;

Returns coordinates for a specific vertex of a link, index = 1..CoordinateExtractCount 66.

For the first and last vertex use function Link2Fromnode 86 / Link2Tonode 86 combined with NodeCoordX 91 / NodeCoordY 91.

*Possible error codes:* -10 -30 -46
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.32 CoordinateExtractCount

### Function CoordinateExtractCount(const link: longint): longint;

Returns how many vertices (or coordinates) a link has, not counting the first and last.

See also Network terminology 26 and coordinateExtract 65.

*Possible error codes:* -10 -46
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.33 CoordinateWindow

### Property CoordinateWindow: double;

This property controls checking of coordinates when entered into functions that accept coordinates.

It will check if coordinates are within the Minimum Bounding Rectangle + X % of the street network.

An example: If the coordinate should be between 0 and 50 and CoordinateWindow=20, then only coordinates between -10 and 60 will be accepted.

By setting CoordinateWindow < 0, the check is disabled. Default value is 20.

This will prevent situations where you by mistake swap x and y coordinate or use lat/long coordinates when the street network was in a projected coordinate system or vice versa.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.34 CoordSys

### Property CoordSys: string;

Returns the coordsys clause (MapInfo format) of the currently loaded network. Read-only.

See also Epsg 68

*ActiveX / VCL / CLX component:* RWnetBase

## 4.35 CostDist

### Property CostDist: double;

See Route for a description. Default value is 1. CostDist>=0.

*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.36 CostTime

### Property CostTime: double;

See Route for a description. Default value is 0. CostTime>=0.

*ActiveX / VCL / CLX component:* RWcalc

## 4.37 Direction1

### Function Direction1(node1, node2, node3: integer): integer;

Returns the turning angle (0-359) at node2 when moving from node1 to node3 via node2. This is based on the coordinates of the nodes and doesn't take the actual polyline-layout into consideration.

Straight on is 0, to the left is 90, backward is 180 and to the right is 270.

See also Direction2 67.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.38 Direction2

### Function Direction2(link1, node2, link2: integer): integer;

Returns the turning angle (0-359) at node2 when moving from link1 to link2 via node2. This is based on the exact coordinates of the polylines and the node.

Link1 and link2 must both be connected to node2. Specifying node2 may seem superfluous, but is required since link1 and link2 could be parallel links. If your network only consists of lines or polylines with one segment, Direction1 gives the same result.

Straight on is 0, to the left is 90, backward is 180 and to the right is 270.

See also Direction1 67.

*Possible error codes:* -10 -30 -31 -40 -43 -46
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.39 Directory

### Property Directory: string;

Directory defines the location of all binary files used by RW Net. Default directory is the current path.

*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.40 Epsg

### Property Epsg: integer

Set this property if you use GML as output format 74. Epsg codes define the coordinate system used by the GML file. It is the developers own responsibility to set this correctly. It is not stored in the binary network files.

Default value is 4326, which means "Latitude / Longitude WGS84".

See also CoordSys 66 and www.epsg.org

*ActiveX / VCL / CLX component:* RWnetBase

## 4.41 ErrorMsg

### Function ErrorMsg(code: integer): string

Translates an error code into a text message. This should not be used in all end-user environments.
See also the list of error codes 49.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.42 ExtendedDataCheck

### Property ExtendedDataCheck: integer;

If this property is a positive number >0, function NWcreate 94 will perform additional checks of the input data during network creation. This includes check for loop links, double-vertices, self-intersections and sharp turns. This is all reported in the file network_report.txt.

The value of the property determines how sharp the turns should be in order to be reported. A value of 10 will for instance report all turns almost U-turn like. 90 means all turns of 90 degrees or more will be reported. So the smaller value, the sharper the turns has to be, to be reported.

Please note that this is only for turns within a single link and is not related to the turns reported as part of driving directions.

If double-vertices are found, no checks are done for self-intersections and sharp turns.

List of possible return codes:

1: Object is marked for deletion / not geocoded
2: Object is not a polyline
3: Object has >1 parts
4: Object has zero length
5: Object is a loop link
6: Object has duplicate vertices (may mean self-intersecting)

© 2013 RouteWare / Uffe Kousgaard

7: Object is self-intersecting
8: Object has a very sharp turn
9: Object has only 0 or 1 vertex

*ActiveX / VCL / CLX component:* RWnetBase

## 4.43 ExternIDfindID

### Function ExternIDfindID(index: integer): string;

Look up the external ID based on the rowID (as returned by function RouteGetLink etc). Remember to call ExternIDimport 69 first.

If return string is empty, an error occurred.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.44 ExternIDfindIndex

### Function ExternIDfindIndex(id: string): integer;

Look up the rowID based on the external ID (as needed for input by function SetLinkSpeed etc). Remember to call ExternIDimport 69 first.

If rowID is <=0, an error occured.

*Possible error codes:* -19 -32 -40
*ActiveX / VCL / CLX component:* RWnetBase

## 4.45 ExternIDimport

### Function ExternIDimport(filename: string): integer

This function creates a lookup table, which can be used for translating between external ID (numbers or strings) and internal rowID (1,2,3..).

Supply a text file with one external ID on each line and call the function. It will automatically detect if numbers or strings are used and create the index most efficiently according to this. Use a fully qualified filename including path.

The index is kept in a file called index.bin in the same directory as the other network files (as defined by property directory 67)

The length of the external ID can not exceed 100 characters.

See also ExternIDimport2 70

*Possible error codes:* -32 -40
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.46 ExternIDimport2

### Function ExternIDimport2(filename,fieldname: string; fieldindex: integer): integer

Same as ExternIDimport 69 except the datasource is a DBF/DAT file.

The file should be of type DBF (version 3) or DAT file (part of a TAB file).

In the case of a DAT file, the corresponding TAB file needs to be present.
If you supply fieldindex=0, the fieldname will be used. If you supply a fieldindex (1..) it will be used for reading from the file.

Field type should be numerical, integer, small integer or character.

Filename should include a fully qualified path.

*Possible error codes:* -20 -22 -30 -32 -40
*ActiveX / VCL / CLX component:* RWnetBase

## 4.47 ExternIDopen

### Function ExternIDopen(cache: boolean): integer;

Opens the index created by ExternIDimport. If cache is true, the whole index will be loaded into memory for faster processing.

*Possible error codes:* -32 -40 -41
*ActiveX / VCL / CLX component:* RWnetBase

## 4.48 ExtraDist

### Property ExtraDist: double;

See Route 106 for a description. Default value is 0.

*ActiveX / VCL / CLX component:* RWcalc

## 4.49 ExtraTime

### Property ExtraTime: double;

See Route 106 for a description. Default value is 1.

*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.50 ExtraVarCreate

### Function ExtraVarCreate: integer;

Creates an extra variable which is updated by function Route 106 and IsoCost 75. Uses 4 bytes per link.

*Possible error codes:* -40 -41
*ActiveX / VCL / CLX component:* RWcalc

## 4.51 ExtraVarDrop

### Procedure ExtraVarDrop;

Frees memory associated with extra variable. This is automatically done, when the network is unloaded.

*ActiveX / VCL / CLX component:* RWcalc

## 4.52 ExtraVarExist

### Function ExtraVarExist: boolean;

Returns true if extra route information has been created.

*ActiveX / VCL / CLX component:* RWcalc

## 4.53 GetLinkCost

### Function GetLinkCost(Link: integer): single;

Same as GetNodeCost 73, just for links instead.

TurnMode 124=false:
Returns the maximum cost of the two end nodes of the link.

TurnMode 124=true:
Link>0: Returns the cost of going to the ToNode
Link<0: Returns the cost of going to the FromNode

*Possible error codes:* -10 -30
*ActiveX / VCL / CLX component:* RWcalc

## 4.54 GetLinkCostDyn

### Function GetLinkCostDyn(Link: integer; percent: double): single;

Returns the cost of getting to a specific location of a link. This function can be used after a call to either IsoCost 75 or IsoCostDyn 75 function.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro

© 2013 RouteWare / Uffe Kousgaard

*ActiveX / VCL / CLX component:* RWcalc

## 4.55 GetLinkCostDyn2

### Function GetLinkCostDyn2(Link: integer; percent: double; var extra: single; approach: integer): single;

Returns the cost and "extra" cost of getting to a specific location of a link. This function can be used after a call to IsoCostDyn2 76 or IsoCostDynLocationList2 76 function.

If link is the same value as when calling IsoCostDyn2, you will get a -30 error. Call RouteDyn_Approach 107 instead.

*Possible error codes:* -10 -18 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.56 GetLinkDist

### Function GetLinkDist(Link: integer): single;

Returns the length of a specific link. The result is returned in km or miles according to property units 126.

If 0 is returned, the corresponding record was deleted or not a line-object in the GIS file used when creating the binary network.

Some links may actually have a length of 0, if it consists of 2 vertices with the same set of coordinates. Such are however reported in file "network_report.txt" as part of the network creation (see NWcreate 94).

*Possible error codes:* -10 -30
*ActiveX / VCL / CLX component:* RWnetBase

## 4.57 GetLinkExtra

### Function GetLinkExtra(Link: integer): single;

Same as GetLinkCost 71, but returns link extra-result.

*Possible error codes:* -10 -30 -39
*ActiveX / VCL / CLX component:* RWcalc

## 4.58 GetLinkExtraDyn

### Function GetLinkExtraDyn(Link: integer; percent: double): single;

Returns the extra variable connected to a specific location of a link. This function can be used after a call to either IsoCost 75 or IsoCostDyn 75 function.

© 2013 RouteWare / Uffe Kousgaard

*Possible error codes:* -10 -30 -39 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.59 GetLinkSpeed

### Function GetLinkSpeed(link: integer): single;

Returns the travelling speed for a specific link. Returned in km/h or miles/h according to property units 126.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.60 GetLinkTime

### Function GetLinkTime(link: integer): single;

Returns the travelling time for a specific link.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.61 GetNodeCost

### Function GetNodeCost(Node: integer): single;

After a call to IsoCost 75 the cost to all nodes are calculated. With GetNodeCost, you can query the cost to individual nodes very fast (instantaneous). This can be used to create a complete distance table.

Please note, if you have called IsoCost with a limit on the distance, the result for some nodes may be unlimited.

This can also be used after a call to Route, but in this case even fewer links will have a defined result. However all nodes on the route are defined.

When used in combination with turn restrictions, remember that a certain node can be part of a route more than once. In this case use GetLinkCost function instead when querying cost along the route.

*Possible error codes:* -10 -30 -43
*ActiveX / VCL / CLX component:* RWcalc

## 4.62 GetNodeExtra

### Function GetNodeExtra(Node: integer): single;

Same as GetNodeCost, but returns extra-result.

© 2013 RouteWare / Uffe Kousgaard

*Possible error codes:* -10 -30 -39
*ActiveX / VCL / CLX component:* RWcalc

## 4.63 GetOpenStatus

### Function GetOpenStatus(link: integer): integer;

Returns the status of the specified link for the current mode:

0: Two-way street
512: One-way street, which may only be travelled in the digitised direction.
1024: One-way street, which may only be travelled in the reverse of the digitised direction.
1536: Closed street.

See also CloseLink 60

*Possible error codes:* -10 -30
*ActiveX / VCL / CLX component:* RWnetBase

## 4.64 GISformat

### Property GISformat;

Defines the output format for functions such as IsoPoly2:

0: MIF (default)
1: SHP
2: MITAB - TAB format (not available in .NET version, will generate MIF file instead)
3: GML2 - GML 2.1.2
4: None
6: KML2 - KML 2.0

TAB format is only possible on the windows platform if the MITAB.DLL is found on the path. If you specify TAB as format and this is not the situation, MIF files will be generated instead. This requires MITAB.DLL version 1.5.0 found at RouteWare website.

MIF and TAB files are generated with coordinate clauses as specified, when the network was created. For SHP files no .PRJ file is created.

GML format will create 2 files with XML and XSD extensions.

KML files can be opened with Google Earth desktop application. You should only use this format if your coordinate data are in latitude/longitude (WGS 84) format, since no coordinate conversion will be performed.

If you plan to read the DBF part of a SHP file, be aware that some database drivers only accepts short (8+3) filenames. Microsofts JET engine has this problem.

This property is available on both objects (except for the DLL version).

If you specify it for RWcalc object, it will override the setting of the RWnetBase

© 2013 RouteWare / Uffe Kousgaard

object. Setting None is only possible for the RWcalc object and is also the default.

*ActiveX / VCL / CLX component:* RWnetBase & RWcalc

## 4.65 Identifier

### Function Identifier: integer;

Returns the ID of a RWnetBaseX object (same as the windows handle).

See also method SetNet 117.

*Versions:* All ActiveX versions.
*ActiveX component:* RWnetBase

## 4.66 Init

### Procedure Init(password: string);

See separate discussion for description 29.

*ActiveX component:* RWnetBase

## 4.67 IsoCost

### Function IsoCost(Node: integer; MaxCost: single): integer;

Calculates an isochrone from node until a maximum cost of maxcost has been reached. If maxcost is 0, there is no upper limit and the whole network will be calculated, except for subnets.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.68 IsoCostDyn

### Function IsoCostDyn(link: integer; percent: double; MaxCost: single): integer;

Calculates an isochrone from the location defined by (link,percent) until a maximum cost of maxcost has been reached. If maxcost is 0, there is no upper limit and the whole network will be calculated, except for subnets.

See Network terminology 26 for a description of a location.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -30 -40 -43
*Versions:* Standard Pro

© 2013 RouteWare / Uffe Kousgaard

*ActiveX / VCL / CLX component:* RWcalc

## 4.69 IsoCostDyn2

### Function IsoCostDyn2(link: integer; percent: double; MaxCost: single; approach: integer): integer;

This is the same function as IsoCostDyn 75, except it takes an additional parameter, approach. This makes it possible to direct the isochrone in a specific direction from the starting point. Use it together with GetLinkCostDyn2 72.

If approach=512 the route will start in the same direction as digitization, if 1024 the reverse.

The main purpose is to calculate full cost matrices for use with curb approach optimizations. See sample 19.

Only possible to use, when turnmode 124 is true.

*Possible error codes:* -10 -18 -30 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.70 IsoCostDynLocationList

### Function IsoCostDynLocationList(link: integer; percent: double; locationnum: integer): integer;

Calculates an isochrone from a location until all locations on the list has been reached. If some of the locations are in a subnet the whole network will be calculated.

Link is the starting link.
Percent is the position along the starting link.
Locationnum denotes the number of locations on the list. See function LocationListSet 88 for more information.

This function is good for calculating distance matrices as fast as possible.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -30 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.71 IsoCostDynLocationList2

### Function IsoCostDynLocationList2(link: integer; percent: double; locationnum, approach: integer): integer;

This is the same function as IsoCostDynLocationList 76, except it takes an additional parameter, approach. This makes it possible to direct the isochrone in a specific direction from the starting point. Use it together with GetLinkCostDyn2 72.

© 2013 RouteWare / Uffe Kousgaard

If approach=512 the route will start in the same direction as digitization, if 1024 the reverse.

The main purpose is to calculate full cost matrices for use with curb approach optimizations. See sample 19.

Only possible to use, when turnmode 124 is true.

*Possible error codes:* -10 -18 -30 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.72 IsoCostDynLocationListN

### Function IsoCostDynLocationListN(link: integer; percent: double; locationnum,nearest: integer; maxcost: single): integer;

Calculates an isochrone from a location until "Nearest" locations on the list has been reached. If maxcost<>0, the function will however stop as soon as maxcost has been reached.

Link is the starting link.
Percent is the position along the starting link.
Locationnum denotes the number of locations on the list. See function LocationListSet 88 for more information.

The function returns the actual number of locations found on the list. After successful return the index's for the location list are updated to reflect the order of the nearest locations. This is done through LocationListGetNewPos 87 and LocationListGetOldPos 88 functions.

This function is good for locating the nearest N objects on the location list as fast as possible. See sample code 14.

You can improve performance of this function by not having any very long links in your network, such as a long ferry route.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -30 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.73 IsoCostMulti

### Function IsoCostMulti(Nodenum: smallint; maxcost: single): integer;

Calculates isochrones simultaneously for several nodes, which has been entered through function NodeListSet 92.

Nodenum defines the number of nodes on the node list and maxcost defines if there is

© 2013 RouteWare / Uffe Kousgaard

a maximum cost. 0 means no maximum.

The ID of the nearest node for each link can be extracted with function ResultSave 101 / ResultGet 101 afterwards.

This function is ideal for locating areas, which has more than x km to a facility, if you have several facilities or testing how new facilities will affect the market.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -40 -41 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.74 IsoCostNodeList

### Function IsoCostNodeList(Node,nodenum: integer): integer;

Calculates an isochrone from node until all nodes on the list has been reached. If some of the nodes are in a subnet the whole network will be calculated.

Nodenum denotes the number of nodes on the list. See function NodeListSet 92 for more information.

This function is good for calculating distance matrices as fast as possible.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -30 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.75 IsoCostNodeListN

### Function IsoCostNodeListN(Node,nodenum,nearest: integer; maxcost: single): integer;

Calculates an isochrone from node until "Nearest" nodes on the list has been reached. If maxcost<>0, the function will however stop as soon as maxcost has been reached.

Nodenum denotes the number of nodes on the list. See function NodeListSet 92 for more information.

The function returns the actual number of nodes found on the list. After successful return the index's for the node list are updated to reflect the order of the nearest nodes. This is done through NodeListGetNewPos 91 and NodeListGetOldPos 92 functions.

This function is good for locating the nearest N objects on the node list as fast as possible. See sample code 29.

Use OnIsoChroneProgress event for tracking progress.

© 2013 RouteWare / Uffe Kousgaard

*Possible error codes:* -10 -30 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.76 IsoCostOffSet

### Property IsoCostOffSet: boolean;

Setting this property to true allows you to add a cost offset to each node, when calculating multi-centered isochrones. This affects these functions: isocostmulti 77, isogrid 80, isolink2 80, isolink4 81, isopoly2 81, isopoly3 83 and isopoly4 84.

This can for instance be used to create drivetime regions around a number of fire-stations, which has different start times.

To enter the offset values, use the ViaListSet 127 functions as in this example:

```
IsoCostOffSet = true
SetFastest
NodeListSet(1,80)
NodeListSet(2,107)
NodeListSet(3,45)
ViaListSet(1,"",2)
ViaListSet(2,"",6)
ViaListSet(3,"",4)
StepsAdd(8)
StepsAdd(12)
IsoPoly2("new_drivetime",3,true,0,0,0,0)
```

This will add 2 minutes to node 80, 6 minutes to node 107 and 4 minutes to node 45. Drivetime isochrones will be drawn at 8 and 12 minutes.

*ActiveX / VCL / CLX component:* RWcalc

## 4.77 IsoCostSave

### Function IsoCostSave(link, node, complete: boolean): integer;

Saves the result from a previous call to the IsoCost 75 function.

Only results>0 are saved unless complete is true. This makes it possible to have more than one isochrone in the network at the same time, unless they are overlapping.

Specify if both link and/or node data is to be saved (true/false).

See also ResultFiles 101 for further details.

*Possible error codes:* -10 -20 -21 -32 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.78 IsoGrid

### Function IsoGrid(nodenum: smallint; Xlong1, Ylat1, Xlong2, Ylat2: single; gridV, gridH: integer; roundoff: single): integer;

IsoGrid calculates an isochrone grid based on 1 or more nodes entered through the standard NodeListSet 92 functions.

The grid is created as a window defined by the coordinates in the parameter list. The number of grids can be defined vertically and horizontally. Grids with same value are joined vertically before they are saved to "isogrid.mif".

It is also possible to define a round-off parameter, which is applied to the cost before the grids are joined. This makes the resulting grid more compact. 10 rounds off to the nearest 10 etc.

*Possible error codes:* -10 -30 -32 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.79 IsoLink2

### Function IsoLink2(filename: string; nodenum: integer): integer;

IsoLink2 calculates isocost links, which shows how far it is possible to go within a specified amount of cost from one or more nodes.

Use procedure StepsClear 119 and StepsAdd 119 to add a list of the actual isochrones generated.

The output is similar to IsoPoly2 81, but the result is a polyline theme instead of a polygon theme and the polylines are dynamically segmented to show the exact position of the steps.

The result is saved to filename, which should include path and filename, but exclude extension.

Nodenum specifies the number of nodes entered into the standard NodeListSet 92.

The output format is determined by property GISformat 74.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -30 -40 -41 -46
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.80 IsoLink2Dyn

### Function IsoLink2Dyn(filename: string; link: integer; percent: double): integer;

© 2013 RouteWare / Uffe Kousgaard

IsoLink2Dyn works the same way as IsoLink2 80 except for this difference:

Starting point is a single location, identified by a link/percent pair.

*Possible error codes:* -10 -30 -40 -41 -46
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.81 IsoLink4

### Function IsoLink4(filename: string; nodenum: integer): integer;

IsoLink4 calculates links, which shows which center is the nearest on a street network.

The functionality is similar to IsoPoly4 84, but the result is a polyline theme instead of a polygon theme and the polylines are dynamically segmented to show the exact position where it changes, which center is the nearest.

The result is saved to filename, which should include path and filename, but exclude extension.

Nodenum specifies the number of nodes entered into the standard nodelist 92 (centers).

The output format is determined by property GISformat 74.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -30 -40 -41 -46
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.82 IsoPoly2

### Function IsoPoly2(filename: string; nodenum: integer; Doughnut: boolean; Xlong1,Ylat1,Xlong2,Ylat2: double): integer;

IsoPoly2 calculates isocost polygons, which shows how far it is possible to go within a specified amount of cost from one or more nodes.

The result is saved to filename, which should include path and filename, but exclude extension.

The output format is determined by property GISformat 74.

Nodenum specifies the number of nodes entered into the standard nodelist 92.

It is possible to define, that doughnut polygons should be generated, e.g 0-5 minutes, 5-10 minutes etc. or standard polygons e.g. 0-5 minutes, 0-10 minutes etc.

Use procedure StepsClear 119 and StepsAdd 119 to add a list of the actual isochrones generated.

© 2013 RouteWare / Uffe Kousgaard

The best way to understand these parameters is to try rwnetdemo.exe, which makes it possible to adjust the parameters easily and see the result immediately.

It is possible to define a bounding box for the calculation. If it doesn't cover at least all nodes in the nodelist, it is extended to do so. Specifying all 0's means it just covers the whole network. See also IsoPoly2Fast 82.

See also property AddNodes 57 and separate discussion 37.

Latitude/longitude coordinates are not really supported by this function, but only in rare situations will it actually affect the generated polygons.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -30 -40 -41 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.83 IsoPoly2Fast

### Function IsoPoly2Fast(filename: string; nodenum: integer; Doughnut: boolean; buffer: double): integer;

This is the same function as IsoPoly2 81, except the required bounding box gets calculated for the user with a buffer added. We recommend a value of 3 km / 2 miles in urban areas and more in rural areas (>10).

This means the calculation goes much faster for small isochrones in large networks and the RAM requirement is also greatly reduced.

There is a small risk of not getting the exact same result as calling IsoPoly2 with 0,0,0,0 as parameter, but this should only occur in extreme situations. If so, increase the size of the buffer.

## 4.84 IsoPoly2Dyn

### Function IsoPoly2Dyn(filename: string; link: integer; percent: double; Doughnut: boolean; Xlong1,Ylat1,Xlong2,Ylat2: double): integer;

IsoPoly2Dyn works the same way as IsoPoly2 81 except for this difference:

Starting point is a single location, identified by a link/percent pair.

*Possible error codes:* -10 -30 -40 -41 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.85 IsoPoly2DynFast

### Function IsoPoly2DynFast(filename: string; link: integer; percent: double; Doughnut: boolean; buffer: double): integer;

This is the same function as IsoPoly2Dyn 82, except the required bounding box gets calculated for the user with a buffer added. We recommend a value of 3 km / 2 miles in urban areas and more in rural areas (>10).

This means the calculation goes much faster for small isochrones in large networks and the RAM requirement is also greatly reduced.

There is a small risk of not getting the exact same result as calling IsoPoly2Dyn with 0,0,0,0 as parameter, but this should only occur in extreme situations.

## 4.86 IsoPoly3

### Function IsoPoly3(filename: string; maxcost: single; nodenum, min_valency: integer; Xlong1,Ylat1,Xlong2,Ylat2: double): integer;

IsoPoly3 calculates voronoi polygons (cells), which shows the cost associated with each node in the network.

The result is saved to filename, which should include path and filename, but exclude extension.

The attribute file also holds information on the id of the nearest node.

The output format is determined by property GISformat 74.

The function can use one or more nodes as a basis for the calculation of node cost. Nodenum specifies the number of nodes entered into the standard nodelist.

Maxcost specifies the maximum cost for the voronoi cells.

By specifying min_valency it is possible to restrict the generation of cells to nodes with a minimum valency, that is the more important nodes.

It is also possible to restrict the cells to a specific area of the network based on a window defined by (xmin,ymin) - (xmax,ymax).

The best way to understand these parameters is to try rwnetdemo.exe, which makes it possible to adjust all parameters easily and see the result immediately.

See also property AddNodes 57 and separate discussion 37.

Latitude/longitude coordinates are not really supported by this function, but only in rare situations will it actually affect the generated polygons.

Use OnIsoChroneProgress event for tracking progress.

© 2013 RouteWare / Uffe Kousgaard

*Possible error codes:* -10 -30 -40 -41 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.87 IsoPoly4

### Function IsoPoly4(filename: string; maxcost: single; nodenum: integer; Xlong1,Ylat1,Xlong2,Ylat2: double): integer;

IsoPoly4 calculates polygons, which shows which areas are closest to each center as defined in the NodeListSet function.

The result is saved to filename, which should include path and filename, but exclude extension.

The output format is determined by property GISformat 74.

The output file holds information on the id of the nearest node.

The function can use one or more nodes as a basis for the calculation of node cost. Nodenum specifies the number of nodes entered into the standard nodelist.

The best way to understand these parameters is to try rwnetdemo.exe, which makes it possible to see the result immediately.

Set maxcost = 0, unless:
1) You want to limit the generated polygons to a maximum distance, so that areas beyond doesn't get related to any center.
or
2) You have a very large network and want to reduce calculation time. Set maxcost to a value, that guarantees all areas are covered.

It is possible to define a bounding box for the generated polygons. If it doesn't cover at least all nodes in the nodelist, it is extended to do so. Specifying all 0's means it just covers the network.

See also property AddNodes 57 and separate discussion 37.

Latitude/longitude coordinates are not really supported by this function, but only in rare situations will it actually affect the generated polygons.

Use OnIsoChroneProgress event for tracking progress.

*Possible error codes:* -10 -30 -40 -41 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.88 LimitCreate

### Function LimitCreate(FileID: integer; filename: string): integer;

This function performs the same operation as LimitCreate2 85, but creates the limit file from a text file instead. This is an alternative function, if your data are not in a

© 2013 RouteWare / Uffe Kousgaard

DBF/TAB format.

The text file shouldn't contain any other values but the attribute, that is one positive integer on each line in the file.

Filename should be without path information, since that is obtained from property Directory 67.

*Possible error codes:* -32 -40
*ActiveX / VCL / CLX component:* RWnetBase

## 4.89 LimitCreate2

### Function LimitCreate2(FileID: integer; filename, fieldname: string; fieldindex: integer): integer;

From the specified filename a file is created, which holds information about limits on links. See description here: Limits 42

FileID is a number from 1-9 and defines the output filename: Output will be a file named "limit1.bin" - "limit9.bin".

A maximum of 9 such restrictions can be created in files and a maximum of 9 of these can be loaded into memory at a time.

The file should be of type DBF (version 3) or DAT file (part of a TAB file).

In the case of a DAT file, the corresponding TAB file needs to be present.
If you supply fieldindex=0, the fieldname will be used. If you supply a fieldindex (1..) it will be used for reading from the file.
If the field, you are pointing at, is a floating point field, it will be rounded.
If any value is negative or >255, it will be set to 0.

Filename should include a fully qualified path.

The generated file simply holds the values from the input file and so it is possible to write directly to this file if needed. It consists of 8-bit signed integers. You may even create the file from scratch yourself, if that suits you better, but you miss being able to encrypt the file.

**Examples:**
LimitCreate2("roads.dbf","",1)
> Use first field from *roads.dbf* file.

LimitCreate2("roads.dat","maxheight",0)
> Use field *maxheight* from *roads.dat* file. File *roads.tab* needs to be present.

See also LimitCreate 84

*Possible error codes:* -20 -22 -30 -32 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.90 LimitLoad

### Function LimitLoad(FileID, LimitID: integer): integer;

FileID is a number from 1-9 and limitID is a number from 1-9. If FileID=1, a file called "limit1.bin" will be loaded into memory.

LimitID defines which position from 1-9 for the limits. You need to load the positions in sequence, so for instance 3 limits should be loaded into limitID 1, 2 and 3. If you choose 1, 2 and 4, 4 will be ignored since 3 is unused.

See description here: Limits 42

*Possible error codes:* -10 -30 -31 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.91 LimitLoad_bitpattern

### Function LimitLoad_bitpattern(FileID, LimitID: integer): integer;

Same as LimitLoad 86, except limits based on bitpatterns are interpreted differently during route calculation.

See description here: Limits 42

*Possible error codes:* -10 -30 -31 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.92 Link2FromNode

### Function Link2FromNode(link: integer): integer;

Returns the number of the node at the start of the link. This is where digitizing has started.

*Possible error codes:* -10 -30
*ActiveX / VCL / CLX component:* RWnetBase

## 4.93 Link2ToNode

### Function Link2ToNode(link: integer): integer;

Returns the number of the node at the end of the link. This is where digitizing has ended.

*Possible error codes:* -10 -30
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.94 LinkLimit

### Function LinkLimit: integer;

Returns the maximum number of links allowed in the network according to your license. This is 3000 for the Free version, 500,000 in the Standard version and 80,000,000 in the Pro version. Actual limits depend on Operating system, amount of RAM, required functionality etc.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.95 LinkMax

### Function LinkMax: integer;

Return the highest link-number in the currently loaded network, which should equal to the number of links in the corresponding GIS-network.

*Possible error codes:* -10
*ActiveX / VCL / CLX component:* RWnetBase

## 4.96 Location2Coordinate

### Function Location2Coordinate(link: integer; percent: double; var xlong,ylat: double): integer;

Translates a location into a set of coordinates. This makes it the reverse function of Coordinate2location 62.

*Possible error codes:* -10 -30 -46
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.97 LocationListGet

### Function LocationListGet(index: integer; var link: integer; var percent: double): integer;

Reads values from the location list. These are returned as link/percent pairs.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.98 LocationListGetNewPos

### Function LocationListGetNewPos(index: integer): integer;

Can be used in the same way as NodeListGetNewPos 91, just for the location list.

*Possible error codes:* -10 -30 -43

© 2013 RouteWare / Uffe Kousgaard

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.99 LocationListGetOldPos

### Function LocationListGetOldPos(index: integer): integer;

Can be used in the same way as NodeListGetOldPos 92, just for the location list.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.100 LocationListLimit

### Function LocationListLimit: integer

Returns current maximum number of nodes in the location list. The start value is 3000 locations. See function LocationListSet 88 for increasing the length of the list.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.101 LocationListSet

### Function LocationListSet(index, link: integer; percent: double): integer;

Use this function to enter a list of locations.

LocationListSet(1,100,0.1) sets the first location to be link #100, 10% from the beginning. All functions taking the location list as input, expects you to start with position 1 in the list.

If LocationListLimit returns 3000 and you call LocationListSet(3001,link,percent), the length of the list will automatically be increased by 100 more locations every time the current limit is exceeded. If you know you need 10000 locations, then start by calling LocationListSet(10000,link,percent) - that will be slightly more efficient.

NOTE: Not all functions relying on the location list may perform well, if you enter many locations into the list.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.102 Mode

### Property Mode;

Define a mode from 1 to 4. Default is 1.

© 2013 RouteWare / Uffe Kousgaard

If you have multiple modes using the same network, you can define a set of speeds for each mode and define which links can't be used by some of the modes.

See also AttributeCreate 58 and LimitLoad_bitpattern 86

*ActiveX / VCL / CLX component:* RWnetBase

## 4.103 NearestLocation

### Function NearestLocation(link: integer; percent: double; locationnum: integer): integer

Finds the nearest location from a list of locations, calculated from a starting location.

Enter a list of possible nearest locations through the standard location list functions and call the function with these parameters:

Link is the starting link.
Percent is the position along the starting link.
Locationnum is the number of locations on the location list 88.

Returns the ID of the nearest location on location list, that is a number between 1 and locationnum. The real location can then be found with LocationListGet 87. If return value=0, none of the locations where found.

You can improve performance of this function by not having any very long links in your network, such as a long ferry route.

*Possible error codes:* -10 -30 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.104 NearestNode

### Function NearestNode(node,nodenum: integer): integer

Finds the nearest node from a list of nodes, calculated from a starting node.

Enter a list of possible nearest nodes through the standard node list functions and call the function with these parameters:

Node is the starting node.
Nodenum is the number of nodes on the node list 92.

Returns the ID of the nearest node on node list, that is a number between 1 and nodenum. The real node ID can then be found with NodeListGet 91. If return value=0, none of the nodes where found.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.105 NearestOpen

### Function NearestOpen(const node: integer; var NearestNode,NearestLink: integer; var cost: single): integer;

This function takes as input a node and will return the nearest open link, that is one which is not closed for driving in both directions. The nearest node, where it is possible to start a route, is also returned and the cost of getting there.

This can be used when starting a route in a pedestrian-only area, where several links are closed for driving, but has addresses attached.

The function also work in turn-restriction mode. Then both one-way restrictions and any turn-restrictions will be ignored in the search for the nearest open link.

Returns 0 if no errors.

*Possible error codes:* -10 -30 -45
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.106 NearestOpenDyn

### Function NearestOpenDyn(const link: integer; const Percent: double; var NearestNode,NearestLink: integer; var cost: single): integer;

Same as function NearestOpen 90, except the input is a link number and a position along the link.

Observe that 0<PERCENT<1.

*Possible error codes:* -10 -30 -45
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.107 NetworkLength

### Function NetworkLength: double;

Returns the total length of the whole network in km or miles according to property units 126.

*Possible error codes:* -10
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.108 NodeCoordX

### Function NodeCoordX(node: integer): double;

Returns the x-coordinate (or longitude) of node.

*Possible error codes:* -10 -30
*ActiveX / VCL / CLX component:* RWnetBase

## 4.109 NodeCoordY

### Function NodeCoordY(node: integer): double;

Returns the y-coordinate (or latitude) of node.

*Possible error codes:* -10 -30
*ActiveX / VCL / CLX component:* RWnetBase

## 4.110 NodeCreate

### Function NodeCreate: integer

Creates file "Node" on disk, which can be used for locating nodes in the network and viewing node-related results in your GIS. First node in the file has ID=1, second node has ID=2 etc. This has the same function as specifying true for parameter in function NWcreate 94.

The output format is determined by property GISformat 74.

Use the OnAnalyzeProgress event for tracking progress.

*Possible error codes:* -10 -32 -40 -50
*ActiveX / VCL / CLX component:* RWnetBase

## 4.111 NodeListGet

### Function NodeListGet(index: integer): integer;

Reads values from the nodelist.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.112 NodeListGetNewPos

### Function NodeListGetNewPos(index: integer): integer;

Can be used as a supplement to NodeListGet 91. See TSP2 120 for an example on how to use it.

*Possible error codes:* -10 -30 -43

© 2013 RouteWare / Uffe Kousgaard

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.113 NodeListGetOldPos

### Function NodeListGetOldPos(index: integer): integer;

Can be used as a supplement to NodeListGet 91. See TSP2 120 for an example on how to use it.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.114 NodeListLimit

### Function NodeListLimit: integer

Returns current maximum number of nodes in the node list. The start value is 3000 nodes. See function NodeListSet 92 for increasing the length of the list.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.115 NodeListSet

### Function NodeListSet(index, node: integer): integer;

Use this function to enter a list of nodes, which is input to a lot of other functions.

This can e.g. be a list of nodes to visit in optimal order (function TSP2 120).

NodeListSet(1,100) sets the first node to be node #100. All functions taking the nodelist as input, expects you to start with position 1 in the list.

If NodeListLimit returns 3000 and you call NodeListSet(3001,node), the length of the list will automatically be increased by 100 more nodes every time the current limit is exceeded. If you know you need 10000 nodes, then start by calling NodeListSet (10000,node) - that will be slightly more efficient.

NOTE: Not all functions relying upon the nodelist may perform well, if you enter many nodes into the list.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.116 NodeMax

### Function NodeMax: integer;

Return the highest node-number in the currently loaded network.

*Possible error codes:* -10
*ActiveX / VCL / CLX component:* RWnetBase

## 4.117 NW3DNodes

In some datasets several nodes has the same set of coordinates. That is not allowed in RW Net data structure and is here referred to as 3D nodes (more nodes on top of each other).

If you for each link in your dataset has 2 fields with fromnode and tonode ID information, you can save the content of these 2 fields in a space-delimited text file and run this function together with your GIS street network.

The output will be a GIS point theme ("Node_3D"), which shows the position of these situations. Format is as defined in property GISformat 74.

The resulting file can be used in one of 3 ways:

1) To slightly (manually) modify the coordinates in the GIS street database, so the coordinates are not the same anymore.

2) As input to function TurnStandard. This requires that only 2 nodes at a time has the same set of coordinates.

3) Combine the links pairwise. This requires that any other attribute information (such as street name) is the same on both sides of the node.

### 4.117.1 NW3DNodesCGF

### Function NW3DNodesCGF(filename1,filename2: string): integer;

See NW3Dnodes 93 for details.

Filename1: Same filename as for NWcreateCGF 95.

Filename2: Name of text file with fromnode and tonode information (with extension, but without path).

*Possible error codes:* -11 -12 -17 -32 -40 -50
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

### 4.117.2 NW3DNodesMIF

## Function NW3DNodesMIF(filename1,filename2: string): integer;

See NW3Dnodes 93 for details.

Filename1: Same filename as for NWcreateMIF 95.

Filename2: Name of text file with fromnode and tonode information (with extension, but without path).

*Possible error codes:* -11 -12 -32 -40 -50
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

### 4.117.3 NW3DNodesSHP

## Function NW3DNodesSHP(filename1,filename2: string): integer;

See NW3Dnodes 93 for details.

Filename1: Same filename as for NWcreateSHP 96.

Filename2: Name of text file with fromnode and tonode information (with extension, but without path).

*Possible error codes:* -11 -12 -13 -32 -40 -50
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

### 4.117.4 NW3DNodesTAB

## Function NW3DNodesTAB(filename1,filename2: string): integer;

See NW3Dnodes 93 for details.

Filename1: Same filename as for NWcreateTAB 96.

Filename2: Name of text file with fromnode and tonode information (with extension, but without path).

*Possible error codes:* -11 -12 -32 -40 -50
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.118 NWcreate

Networks can be generated from 4 standard GIS formats: TAB, MIF, SHP and CGF.

Generally NWcreate accepts all records where the object type is a polyline with 1 section. All other record types are ignored, but doesn't produce an error:

- Polyline objects with >1 section are ignored (all 4 formats)
- Records marked for deletion are ignored (TAB and CGF format)

© 2013 RouteWare / Uffe Kousgaard

- Records without objects are ignored (all 4 formats)
- Records with other objects than polylines are ignored (TAB, MIF and CGF format)

Even though both deleted records and other kind of objects are supported, you should try to avoid them as the network files get unnecessarily big and count towards to the limit 87.

If you have records marked for deletion in your GIS file, you should be careful when generating data for AttributeCreate. Here you should make sure, that a record for the deleted records are created as well. Best suggestion is to pack the table first.

You can only create a network from 1 GIS file. If you have several GIS files (e.g. one for each county) and you want to create a network for the whole area, you will have to create one large GIS file first.

When generating the network, a text file called "network_report.txt" is created on disk which contains information about the process and if any problems were encountered. See property ExtendedDataCheck 68 for check of additional problems.

The list of generated files can be seen here 29.

Use OnNWcreateProgress event for tracking progress.

See also AttributeCreate2 58, NWcreateTAB 96, NWcreateMIF 95, NWcreateSHP 96, NWcreateCGF 95, SpatialIndexLinksPerCell 119 and ZLevelFile 128.

### 4.118.1 NWcreateCGF

## Function NWcreateCGF(filename: string; nodelayer: boolean; coordsys: string): integer;

The topological network is created from a Cartoworld CGF file (cgf).

See NWcreate 94 for details.

Input: Filename should be without extension and the location of the file is determined by property Directory 67.
If nodelayer=true then function NodeCreate 91 is called at the same time.
The coordinate clause 66 (MapInfo mif/mid format) can be specified. If left empty, a default one is created.

Password protected files are rejected.

*Possible error codes:* -11 -12 -14 -17 -32 -40 -50
*ActiveX / VCL / CLX component:* RWnetBase

### 4.118.2 NWcreateMIF

## Function NWcreateMIF(filename: string; nodelayer: boolean): integer;

The topological network is created from a MapInfo MIF/MID file (mif).

© 2013 RouteWare / Uffe Kousgaard

See NWcreate 94 for details.

Input: Filename should be without extension and the location of the file is determined by property Directory 67.
If nodelayer=true then function NodeCreate 91 is called at the same time.

*Possible error codes:* -11 -12 -14 -32 -40 -50
*ActiveX / VCL / CLX component:* RWnetBase

### 4.118.3 NWcreateSHP

## Function NWcreateSHP(filename: string; nodelayer: boolean; coordsys: string): integer;

The topological network is created from a ESRI SHP file (shp).

See NWcreate 94 for details.

Input: Filename should be without extension and the location of the file is determined by property Directory 67.
If nodelayer=true then function NodeCreate 91 is called at the same time.
The coordinate clause 66 (MapInfo mif/mid format) can be specified. If left empty a default is created.

*Possible error codes:* -11 -12 -14 -32 -40 -50
*ActiveX / VCL / CLX component:* RWnetBase

### 4.118.4 NWcreateTAB

## Function NWcreateTAB(filename: string; nodelayer: boolean): integer;

The topological network is created from a MapInfo TAB file (tab).

See NWcreate 94 for details.

Input: Filename should be without extension and the location of the file is determined by property Directory 67.
If nodelayer=true then function NodeCreate 91 is called at the same time.

If you are using an affine coordinate system, it will not be detected and the coordinate clause will be wrong. Due to the status of the TAB-format (no official description), the network created may be faulty. If you encounter any problems, please report to RouteWare.

*Possible error codes:* -11 -12 -14 -32 -40 -50
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.119 NWload

### Function NWload: integer;

Loads the topological network into memory. Uses app. 34 bytes per link.

For each instance of RWcalc object attached:
Add 9 / 15 bytes per link (turnmode= false / true).
Add 3 / 8 bytes per link if extravar is true.

Maximum is thus 34 + 15 + 8 = 57 bytes per link in the DLL version, which always has exactly one instance of RWcalc.

These numbers are based on average road networks (here TIGER data).

If you also has cached coord3.bin 61, it will of course be higher. The actual file size matches that of the additional RAM required.

It is important that you set property Coord correctly, when you load the network and while you use it.

Use OnNWloadProgress event for tracking progress.

*Possible error codes:* -11 -12 -15 -31 -40 -41
*ActiveX / VCL / CLX component:* RWnetBase

## 4.120 NWloaded

### Function NWloaded: boolean;

Returns true if the network has been loaded.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.121 NWunload

### Procedure NWunload;

Unloads the topological network and frees corresponding memory. All indexes and temporary results are also freed.

It is not needed to call NWunload unless you want to save memory since it is always called automatically, when creating / loading a new network etc. It is also called automatically, when the network object is free'd.

*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.122 OptimumAlpha

### Function OptimumAlpha: single;

Returns the optimum value of alpha. This takes into consideration the relationship between straight line distance and time on all links and value of costdist and costtime.

You only need to call this function and update the Alpha 57 parameter, if you have changed the drive time on some of the links (AttributeLoad 59, SetLinkSpeed 116 or SetLinkTime 117) or changed either property CostDist 66 or CostTime 67 parameter.

If CostTime<>0 this function actually takes some time to process, so don't call it more than needed.

See the discussion on alpha 32 for more information.

*Possible error codes:* -10 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.123 POIadd

### Procedure POIadd(link: integer; percent: double; text: string);

Adds a new POI to the list of POI for use in RouteList 111 function.

Call Coordinate2location 62 to get link / percent from the coordinates of the POI.

See also POIadd2 98.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.124 POIadd2

### Procedure POIadd2(link: integer; percent: double; text: string; direction: integer);

The same function as POIadd 98, except it takes an additional paramter, so it can be controlled in which direction the POI will be included.

Direction:
-1: Include only if link is travelled in opposite of digitized direction
0: Both directions
+1: Include only if link is travelled in digitized direction

There are basically 2 situations:

1) POI that can be seen by all, no matter side of road, direction of vehicle or right / left-hand driving. Use POIadd 98 instead.

© 2013 RouteWare / Uffe Kousgaard

2) POI that can be seen, if the vehicle is on the same side of the road as the POI. The POI is facing towards the traffic.

From the call to Coordinate2location 62 you have the side of the road. Now you can look up the direction parameter in this table:

| Side | Right-hand driving | Left-hand driving |
|---|---|---|
| 0 | +1 | -1 |
| 1 | -1 | +1 |

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.125 POIclear

### Procedure POIclear;

Clears the list of POI for use in RouteList 111 function.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.126 PositionListGet

### Function PositionListGet(index: integer; var x,y: double): integer;

Read coordinates of positions entered into the positionlist. See also PositionListSet 99.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.127 PositionListSet

### Function PositionListSet(index: integer; x,y: double): integer;

Use this function to enter a list of coordinates, which can be used together with function RouteList 111. The positionlist is used when including off-road sections in the graphical output.

The list will automatically grow as you enter more entries.

See also CoordinateWindow 66.

*Possible error codes:* -10 -30 -43 -62
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.128 ReadSpeed

### Function ReadSpeed(filename, fieldname: string; fieldindex: integer): integer;

From the specified filename speed is read for each link. Non-positive speeds are interpreted as the link being closed - the same as calling CloseLink 60 with parameter 1536.

The file should be of type DBF (version 3) or DAT file (part of a TAB file).

In the case of a DAT file, the corresponding TAB file needs to be present.
If you supply fieldindex=0, the fieldname will be used. If you supply a fieldindex (1..) it will be used for reading from the file.

Filename should include a fully qualified path.

*Possible error codes:* -10 -20 -22 -30 -32 -40 -51
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.129 Result

Several functions perform calculations, which as a return value, assigns a result to each link and/or node in the network. These results are stored in an internal array:

Functions, which updates the result array (link):
CulDeSac 135, FindRoundAbout 140, Join 143, IsoCostMulti 77, IsoGrid 80, ParallelLinks 147, SubNet 150 and UnUsedLinks 151.

Functions, which updates the result array (node):
IsoCostMulti 77 and SubNet 150.

The results can be accessed by these RW Net functions:
One item at a time: ResultGetLink 101 / ResultGetNode 101
Saved to file: ResultSaveLink 101 / ResultSaveNode 101.

### 4.129.1 ResultDrop

### Procedure ResultDrop;

Frees memory associated by the result array - see Result 100. This is automatically called, when a network is unloaded.
"4*linkmax+4*nodemax" bytes are freed.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

### 4.129.2 ResultGetLink

## Function ResultGetLink(link: integer): single;

Returns the result for a specific link - see also Result 100.

*Possible error codes:* -10 -30 -37 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

### 4.129.3 ResultGetNode

## Function ResultGetNode(node: integer): single;

Returns the result for a specific node - see also result 100.

*Possible error codes:* -10 -30 -37 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

### 4.129.4 ResultSaveLink

## Function ResultSaveLink: integer;

Saves the link result from a previous calculation to an external file. See Result 100 and ResultFiles 101 for details.

*Possible error codes:* -10 -20 -21 -32 -37 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

### 4.129.5 ResultSaveNode

## Function ResultSaveNode: integer;

Saves the node result from a previous calculation to an external file. See Result 100 and ResultFiles 101 for details.

*Possible error codes:* -10 -20 -21 -32 -37 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.130 ResultFiles

Certain results can be saved directly to a MapInfo TAB (DAT) file or a Shape (DBF) file. This is done by writing directly to the files, which is very fast. This method however requires, that a few issues are taken care of:

1) The output file needs to contain a field of type decimal(11,3)
2) Exclusive access to the file is needed.

Especially 2) can give trouble in MapObjects and ArcGIS for instance, while both MapInfo and ArcView 3 closes all files, while not re-drawing.

© 2013 RouteWare / Uffe Kousgaard

If it doesn't work, use methods ResultGetLink 101 / ResultGetNode 101 instead of the ResultSave functions.

Output from RW Net can be done directly to result files with these methods: IsoCostSave 79, ResultSaveLink 101, ResultSaveNode 101 and RouteSave 115.

### 4.130.1 ResultField

## Property ResultField;

ResultField sets the name of the field into which results 100 are saved. Default ResultField is "result". This value is used for both saving link and node results.

See also ResultFiles 101 for details.

*ActiveX / VCL / CLX component:* RWcalc

### 4.130.2 ResultFileLink

## Property ResultFileLink;

Name of a MapInfo dat-file or a shape dbf-file with a field named as defined in ResultField 102 of type decimal(11,3). The path is defined through Directory 67. The file need to have as many records as there are links in the street network.

See ResultFiles 101 for details.

*ActiveX / VCL / CLX component:* RWcalc

### 4.130.3 ResultFileNode

## Property ResultFileNode;

Name of a MapInfo dat-file or a shape dbf-file with a field named as defined in ResultField 102 of type decimal(11,3). The path is defined through Directory 67. The file need to have as many records as there are nodes in the street network.

See ResultFiles 101 for details.

*ActiveX / VCL / CLX component:* RWcalc

### 4.130.4 ResultFileClear

## Function ResultFileClear(link,node: boolean): integer;

Clears the result file. The result field is assigned with "0.000".
State if one or both result files should be cleared.

See ResultFiles 101 for details.

*Possible error codes:* -20 -21 -32
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.131 RoadNameCreate

### Function RoadNameCreate(ID: integer; filename: string): integer;

The roadname functions is a small database built into RW Net, that will allow you to store a list of road names (or any text for that matter) associated with each link in the network. Since most road names are used on several links, RW Net creates a list of possible names and assigns a lookup value (roadnameID) for each link. This allows you to compare 2 road names by just comparing roadname ID's, which is much faster and requires less RAM.

It is possible to load 99 different roadname database at one time.

**Example on general use**

```
RoadNameCreate(1,"roadname.txt")
RoadNameLoad(1,false)
RoadName1_Get(1,100,name)
Print "Link 100: "+name
RoadNameClose(1)
```

This function will import a text file with road names (1 string per line) and store it as a binary file on disk. There should be as many lines in the text file as you have links in your network.

Filename should include a fully qualified path. ID is an integer from 1 - 99.

The 2 generated files will have names roadnameXX.bin and roadnumberXX.bin, where XX = the ID parameter.

This means you can have several such pair of files - for instance based on different languages or other kind of setups, where further information is included: "Main Street" or "Main Street (Smalltown)".

The text format should be ASCII, ANSI or UTF-8. If you plan to generate GML output, you should use UTF-8 format. Basic principle is: What comes in, also comes out as no conversion is done inside RW Net.

Unicode / UTF-16 is not supported.

Use OnRoadNameCreateProgress event for tracking progress.

See also RoadNameCreate2 103

*Possible error codes:* -30 -32 -40 -48
*ActiveX / VCL / CLX component:* RWnetBase

## 4.132 RoadNameCreate2

### Function RoadNameCreate2(ID: integer; filename,fieldname1,fieldname2: string; fieldindex1,fieldindex2: integer): integer;

© 2013 RouteWare / Uffe Kousgaard

This function performs the same operation as RoadNameCreate 103 except the source of input is different:

The file should be of type DBF (version 3) or DAT file (part of a TAB file). In the case of a DAT file, the corresponding TAB file needs to be present.

You can specify 2 fields for defining the roadname. The first one will be used unless the second is different from "".

If you supply a fieldindex (1..) it will primarily be used as identification of the field. If you supply fieldindex=0, the fieldname will be used instead.

Filename should include any needed path information.

**Examples:**
RoadNameCreate2(1,"roads.dbf","roadname","",0,0)
> Use field *roadname* from *roads.dbf* file.

RoadNameCreate2(1,"roads.dat","roadname","routenumber",0,0)
> Use field *roadname* from *roads.dat* file unless field *routenumber* holds a non-empty value. File *roads.tab* needs to be present.

*Possible error codes:* -20 -22 -30 -32 -40 -48
*ActiveX / VCL / CLX component:* RWnetBase

## 4.133 RoadNameLoad

### Function RoadNameLoad(ID: integer; cache: boolean): integer;

This function will open a pair of roadname files with the defined ID and either just read the roadnumberXX.bin file into memory (cache=false) or read both of them into memory (cache=true).

*Possible error codes:* -10 -30 -31 -51
*ActiveX / VCL / CLX component:* RWnetBase

## 4.134 RoadName1_Get

### Function RoadName1_Get(ID,linkID: integer; var text: string): integer;

This function will return the road name (in text variable) for a specific pair of files (defined by ID) and linkID.

*Possible error codes:* -10 -30 -49
*ActiveX / VCL / CLX component:* RWnetBase

## 4.135 RoadName2_Get

### Function RoadName2_Get(ID,roadnameID: integer; var text: string): integer;

© 2013 RouteWare / Uffe Kousgaard

This function will return the road name (in text variable) for a specific pair of files (defined by ID) and roadnameID.

*Possible error codes:* -10 -30 -49
*ActiveX / VCL / CLX component:* RWnetBase

## 4.136 RoadNameID_Get

### Function RoadNameID_Get(ID,linkID: integer): integer;

This function will return the roadnameID for a specific pair of files (defined by ID) and linkID.

*Possible error codes:* -10 -30 -49
*ActiveX / VCL / CLX component:* RWnetBase

## 4.137 RoadNameClose

### Procedure RoadNameClose(ID: integer);

This function will close a road name file that has previously been opened.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.138 RoundAbout

### Function RoundAbout(link: integer): boolean;

Returns true if a link is part of a roundabout. In all other situations false is returned. This is defined through attribute codes 27.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.139 RoundAboutExitNode

### Function RoundAboutExitNode(node: integer): boolean;

Returns true, when
- The valency of the node is >=3 and
- Two of the links connected to the node is marked as a roundabout and
- It is possible to leave the roundabout using the third link

Returns false otherwise.

This is used internally by function RouteList 111 for creating driving directions.

*Possible error codes:* -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.140 Route

### Function Route(Node1,Node2: integer): single;

Calculates the cost of the cheapest route from node1 to node2 according to Alpha 32, CostTime 67 and CostDist 66. Returns the Cost.

Cheapest route is defined as the route, which minimizes this expression:

Cost = CostDist*distance + CostTime*time, where CostDist and CostTime >=0.

Either CostDist or CostTime should be >0.

If ExtraVarCreate 71 has been called, an additional variable is calculated as

extra = extradist*distance + extratime*time.

Distance is defined according to property units 126 and time is always minutes.

If error -33 is returned, you can check function BestNode 60 to see the node, which was nearest to node2.

Use OnRouteProgress event to track progress in very large networks. In small / medium sized networks it is not needed.

*Possible error codes:* -10 -30 -33
*ActiveX / VCL / CLX component:* RWcalc

## 4.141 RouteDyn

### Function RouteDyn(link1,link2: integer; percent1,percent2: double; var fromto1,fromto2,routelength: integer; var extra: single): single;

This function is used for dynamic routing. This means a route can be calculated from somewhere along a link to somewhere along another link. The normal Route function always goes from node to node.

Link1 and percent1 denotes where on link1 the route should start. The percentage (0 < percent1 < 1) counts from the same end as the link has been digitized.

Link2 and percent2 is the same, just for the link, where the route ends.

Fromto1 is part of the result. If fromto1=0, the calculated route goes through the start node of link1. If fromto1=1, the calculated route goes through the end node of link1.

Fromto2 has the same meaning, just for link2.

Routelength holds the number of nodes in the calculated route. If routelength=0, the route is a subset of link1. This can only happen if link1=link2, that is if the routing is along the same link. Even when link1=link2, routelength can also be >0.

© 2013 RouteWare / Uffe Kousgaard

You should never call function RouteFind 110 after RouteDyn as this is an integrated part of RouteDyn. However RouteGetLink 110, RouteGetNode 111 etc. can be called the normal way.

Variable extra hold the extra cost, if ExtraVarCreate has been called.

The result holds the cost of the route.

There are a few restrictions when using loop links, that is links which start and end at the same node:
Loop links can not hold one-way information.
When computing a route where link1=link2 and the link is also a loop link, turn restrictions are not considered.

It is advisable to split loop links in the network in 2 sections, this removes the restrictions mentioned above.

Use OnRouteProgress event to track progress in very large networks. In small / medium sized networks it is not needed.

See sample code 17.

*Possible error codes:* -10 -30 -33 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.142 RouteDyn_Approach

**Function RouteDyn_Approach(link1,link2: integer; percent1,percent2: double; approach1,approach2: integer; var fromto1,fromto2,routelength: integer; var extra: single): single;**

Same function as RouteDyn 106, except it offers 2 additional parameters:

It is possible to define how to leave the first link and approach the last link on the route.

This can be used to make sure school-kids are picked up at the correct side of a street or dropped off in front of a school without having to cross the street.

If approach1=512 the route will start in the same direction as digitization, if 1024 the reverse. Use 0 if direction doesn't matter.

If approach2=512 the route will end in the same direction as digitization, if 1024 the reverse. Use 0 if direction doesn't matter.

*Possible error codes:* -10 -30 -33 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.143 RouteFileCreate

### Function RouteFileCreate(filename: string; numfields: integer): integer;

Use this function to create a new GIS polyline file from scratch. The file is generated with the same file type as specified in property GISformat 74 and the filename supplied is the full filename, including directory path, but excluding file extension.

Numfields parameter is a positive number >0 with the number of attribute fields.

This set of functions work at a rather low level, so care should be taken to use the functions properly, or you may experience unusable output files.

The last part of Sample code 14 shows an example on how to use it.

*Possible error codes:* -10 -30 -31 -40
*ActiveX / VCL / CLX component:* RWcalc

## 4.144 RouteFileFieldAdd

### Function RouteFileFieldAdd(fieldname: string; fieldtype,fieldwidth,decimals: integer): integer;

Call this function once for each attribute field defined in function RouteFileCreate 108.

Fieldtype is a number from 1-7:
1: Char
2: Integer (signed 32-bit)
3: Smallint (signed 16-bit)
4: Decimal
5: Float
6: Date
7: Logical

The TAB and MIF format supports all of them, while the SHP format maps type 2, 3 and 5 to type 4.

Fieldwidth and decimals are only relevant for some field types (1 and 4).

*Possible error codes:* -10 -30 -31 -40
*ActiveX / VCL / CLX component:* RWcalc

## 4.145 RouteFileRecordAdd

### Function RouteFileRecordAdd(numsections: integer; attributes: string): integer;

After all fields has been defined, one record at a time can be added. Specify the number of sections and all the attributes as a string.

The attributes string contains the values for all fields in one operation. This is a

© 2013 RouteWare / Uffe Kousgaard

comma-delimited string and if the string holds any floating point numbers these has to be with "." as decimal point. Date fields are supplied as YYYYMMDD. String fields must be enclosed with quotes. If you need a quote inside the string, use a double quote. Logical fields are supplied as true ("T", "t", "Y" or "y") or false ("F", "f", "N" or "n").

If numsections=0, you will get a null object for that record and you don't have to call function RouteFileRecordAddSection 109 afterwards. This is perfectly legal and can be used when creating driving directions, where you want to add lines to the description, but without a geographical part.

*Possible error codes:* -10 -30 -31 -40
*ActiveX / VCL / CLX component:* RWcalc

## 4.146 RouteFileRecordAddCustomSection

### Function RouteFileRecordAddCustomSection (Xlong1,Ylat1,Xlong2,Ylat2: double): integer;

This will add a custom section to the current record. A custom section doesn't have to be a part of the existing street network, but can be anything such as the off-road section from a coordinate to a location.

You should call this function as many times, as you have defined in the call to RouteFileRecordAdd 108, the numsections parameter.

*Possible error codes:* -10 -30 -31 -40
*ActiveX / VCL / CLX component:* RWcalc

## 4.147 RouteFileRecordAddSection

### Function RouteFileRecordAddSection(link: integer; startpercent,endpercent: double): integer;

This will add a section to the current record. A section is here a full link from the street network or perhaps just a smaller part, if you define either startpercent<>0 or endpercent<>1.

You should call this function as many times, as you have defined in the call to RouteFileRecordAdd 108, the numsections parameter.

*Possible error codes:* -10 -30 -31 -40
*ActiveX / VCL / CLX component:* RWcalc

## 4.148 RouteFileClose

### Function RouteFileClose: integer

When you are done adding records to the route file, call this function.

*Possible error codes:* -10 -30 -31 -40
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.149 RouteFind

### Function RouteFind(node2: integer): integer;

Locates the link and nodes on a route from node2 to node1, which has been defined in a previous call to either Route 106 or IsoCost 75.

Returns the number of nodes on the route. The number of links is one less.

See also RouteGetLink 110 and RouteGetNode 111.

*Possible error codes:* -10 -30 -35
*ActiveX / VCL / CLX component:* RWcalc

## 4.150 RouteFindDyn

### Function RouteFindDyn(link: integer; percent: double; var fromto1,fromto2,routelength: integer; var extra: single): single;

This function is used for getting a route to location (link,percent) after a call to IsoCostDyn.

The following 2 examples perform the same calculation (that is calculating 2 routes), but the second method is much faster, when you want to calculate many routes starting at the same location. Depending on your actual data, you may need much more than 2 routes to see an improvement:

RouteDyn(link1,link2a,percent1,percent2a,fromto1a,fromto2a,routelengtha,extraa)
RouteDyn(link1,link2b,percent1,percent2b,fromto1b,fromto2b,routelengthb,extrab)

IsoCostDyn(link1,percent1,0)
RouteFindDyn(link2a,percent2a,fromto1a,fromto2a,routelengtha,extraa)
RouteFindDyn(link2b,percent2b,fromto1b,fromto2b,routelengthb,extrab)

Please see function RouteDyn 106 for further details.

*Possible error codes:* -10 -30 -33 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.151 RouteGetLink

### Function RouteGetLink(index: integer): integer;

After a call to RouteFind 110, you can query the resulting route and get the ID of all links on the route. The ID's are returned in reverse order, so RouteGetLink(1) returns the link closest to the target (i.e. node2).

"ID" is returned if a link is travelled in the digitized direction. If the link is travelled in the opposite direction then "-ID" is returned.

Errors are always returned as "0" !

© 2013 RouteWare / Uffe Kousgaard

*ActiveX / VCL / CLX component:* RWcalc

## 4.152 RouteGetNode

### Function RouteGetNode(index: integer): integer;

After a call to RouteFind 110, you can query the resulting route and get the ID of all nodes on the route. The ID's are returned in reverse order as it is described for function RouteGetLink 110.

*Possible error codes:* -10 -35 -36
*ActiveX / VCL / CLX component:* RWcalc

## 4.153 RouteList

### function RouteList(filename: string; tspmode,listmode,listnum,concatmode: integer; offroadspeed: double; Starttime: Tdatetime; timeformat: integer; vialist: boolean; roadnameID: integer; var distance,time: double): integer;

This function allows you to enter a list of nodes, locations or positions and have a route calculated, that passes through all of them and have the result generated in various ways, ready to be shown on a map. It includes the option of doing travelling salesman optimization and creating driving directions at the same time, making it much easier to get very complex results generated by just specifying a number of parameters.

The parameters are explained here, one by one:

**Output filename**
This is fully qualified filename with path. As usual don't specify the extension as that is determined by the GISformat 74 property.

**TSPmode**
This defines how the routing points are connected. In case of optimization, the TSP2 120 function is used and most of the parameters are also the same as for that function. Only 3 and 4 are new:

0: Optimization, start - end
1: Optimization, round-trip
2: Optimization, start - no fixed end
3: No optimization, start-end
4: No optimization, round-trip
10: Optimization, start - end (straight line distances used in optimization)
11: Optimization, round-trip (straight line distances used in optimization)
12: Optimization, start - no fixed end (straight line distances used in optimization)

**Listmode**
This defines the type of input for the routing points:
1: Nodelist 92

© 2013 RouteWare / Uffe Kousgaard

2: Locationlist 88 (dynamic segmentation)
3: Positionlist 99 (dynamic segmentation)

If you use mode 3, the positions will be translated into locations as part of the function call (and overwrite anything on the Locationlist). In the output you will see a section from the road and to the exact position. This part is referred to as "off-road" and the speed is defined by setting parameter offroadspeed (see below).

**Listnum**
Number of routing points on the list - refers to parameter above.

**Concatmode**
This describes how the GIS objects should be concatenated:
1: With all road segments between routing points as 1 record
2: With off-road segments separately
3: Driving directions
4: With all segments separately

**Offroadspeed**
This is for use with listmode 3. If this is zero and listmode=3, it is effectively the same as listmode=2, but you don't have to call the coordinate2location function yourself, making the setup a bit easier.
The value should be expressed as either km/h or mi/h depending on property units 126 .

**Starttime**
You can use this parameter to define, when the route starts and have a timestamp on all records, which tells when that exact section of the route is entered. Format is a fraction of a day (11:23 = 11/24 + 23/(24*60) = 0.4743055). The integer part of the argument is not used unless timeformat=3.

**Timeformat**
This specifies the format for the timestamp field:
0: Skip timestamp field in output
1: 24 hour format (H:mm)
2: AM/PM (h:mm)
3: Unformatted floating point number - time of day.

At midnight the time wraps for (1) and (2), so "23:59" is followed by "0:00" and "0:01". If you prefer to do your own formatting, use (3).

**Vialist**
This is a list that can contain a name and a time for the routing points to be visited on the route. These will be included in the output and the time will be added to the running total. Specify true / false. See function vialistset 127. If name="" and time=0, the via point will not be included in the output, but will still be used in the routing.

**RoadnameID**
This points to an already loaded database with road names. See RoadName 103 for further info.
You can use 0 as parameter, if no Roadname database has been loaded.

**Distance**
This variable returns the total distance in km or miles.

© 2013 RouteWare / Uffe Kousgaard

**Time**
This variable returns the total time in minutes.

In order for both distance and time to be returned, you need to call ExtraVarCreate 71 in advance.

***These are additional options, not controlled through RouteList parameters:***

**POI in driving directions output**
It is possible to include POI (point-of-interest) in the output, when concatmode = 3. This can be toll stations, petrol stations etc. You can have thousands of POI, but only those on the actual route will be included in the output.

This is an example of output:

| Description | Distance | Total distance |
|---|---|---|
| Road 1 | 0.2 | 0.2 |
| Road 2 | 0.3 | 0.5 |
| POI | 0 | 0.4 |
| Road 3 | 0.5 | 1.0 |

Explanation:
Road 1 is travelled from 0.0-0.2 km
Road 2 is travelled from 0.2-0.5 km
POI is at 0.4 km (on Road 2)
Road 3 is travelled from 0.5-1.0 km

See function POIadd 98 on how to add POI.

**RouteListExitMode**
This parameter defines how exit links from round-abouts are counted. See property RouteListExitMode 114

**SetApproach**
You can define curb approach for the whole route through the seperate function SetApproach 149. This is only available in Pro version.

**SharpTurnDrivingDirections**
This parameter is defined through setting the corresponding property 118. If the value is >0, it is possible to trigger a turn description in the output even when the street name doesn't change, but the road makes a clear turn in an intersection. Just define how sharp the turn should be. Suggested value is 60-75 degrees. Please note this only applies to sharp turns in intersections - not halfway down a link. Such can be reported by setting property ExtendedDataCheck 68.

The routelist function will return 0 if no error occurred.

The possible fields of the generated file are these:
- ID (only if vialist = true)
- LinkID (only when concatmode = 4)
- Roadname or vialist name (only when concatmode = 3 or 4)
- Time (minutes, 1 decimal after the comma = 6 sec accuracy)

© 2013 RouteWare / Uffe Kousgaard

- Total time (minutes, 1 decimal after the comma = 6 sec accuracy)
- Timestamp (string or float format, uses offset value)
- Distance (3 decimals after the comma = meter accuracy)
- Total distance (3 decimals after the comma = meter accuracy)
- Speed (1 decimal after the comma)
- Turn (integer)

Turn is coded this way:
-1: Starting point, if via points are used in output
-2: If next line is a via point
-3: end point, if via points are used in output
0-359: Turning angle
360: Not possible
361-: "Take (value-360). exit from roundabout"

Turning angles from 0-359 can be interpreted this way (45 deg for each section):

0-22: Straight on
23-67: Slight turn to the left
68-112: Turn to the left
113-157: Sharp turn to the left
158-202: U-turn like
203-247: Sharp turn to the right
248-292: Turn to the right
293-337: Slight turn to the right
338-359: Straight on

You are of course welcome to define your own verbal description of turning angles.

Some of the fields may be skipped in the output depending on the input parameters.

See sample code 25.

*Possible error codes:* -10 -30 -33 -38 -40 -49 -55 -56
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.154 RouteListExitMode

### Procedure RouteListExitMode(mode: integer);

This lets you change how exit links are counted in roundabouts in function RouteList 111. By default (value 0) only exit links are counted. If you set it to 1, all links are counted: Entry links and closed links are included.

*ActiveX / VCL / CLX component:* RWcalc

## 4.155 RouteMaxCost

### Function RouteMaxCost(iterations: integer; var node1, node2: integer): single;

This function will return the maximum cost of any cheapest node-2-node route in the

© 2013 RouteWare / Uffe Kousgaard

network. Node1 and node2 will return the nodes for the associated route.

The function uses an approach of a random start points to locate the maximum cost, so a number of iterations needs to be specified. 10 is usually enough, but in rare situations you may need more.

Using dynamic segmentation it may be possible to locate higher costs than for node-2-node routes.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.156 RouteReady

### Function RouteReady: boolean;

Returns true if a call to RouteFind 110 has been made and a route is ready for output.

*ActiveX / VCL / CLX component:* RWcalc

## 4.157 RouteSave

### Function RouteSave(link, node: boolean): integer;

After a call to RouteFind 110 you can save the resulting route directly to either a MapInfo dat-file or a shape dbf-file. Links/nodes not on the route isn't cleared.

State if links and/or nodes should be saved.

Links/nodes on the route is changed to "1", except for start and end-node, which is changed to "2".

See also ResultFiles 101 for further details.

*Possible error codes:* -10 -20 -21 -32 -35
*ActiveX / VCL / CLX component:* RWcalc

## 4.158 SetFastest

### Procedure SetFastest;

This function is a shortcut to defining cost as the fastest route and extra variable as distance. It simply sets these 4 parameters this way:

costtime = 1
costdist = 0
extratime = 0
extradist = 1

See also SetShortest 117

*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.159 SetLimit

### Procedure SetLimit(limitID, value: integer);

Defines the limit for a route or isochrone.

LimitID should be 1-9.
Value should be 0-255. 0 (default) means no limit.

See description here: Limits 42

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.160 SetLinkResult

### Function SetLinkResult(link: integer; value: single): integer;

This function makes it possible to change the result on individual links. This is primary for input to function CPP 132.

*Possible error codes:* -10 -30
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.161 SetLinkSpeed

### Function SetLinkSpeed(link, speed: integer): integer;

Use this function for changing the speed on a specific link. This only affects the speed as it is loaded into memory. After calling AttributeLoad 59 the speed is back to normal, i.e. as defined through the function SetSpeed 118.

This function is of most use for temporarily changing the speed of link in connection with roadwork's etc.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.162 SetLinkSpeedDefault

### Function SetLinkSpeedDefault(link: integer): integer;

Use this function for setting the speed on a specific link back to the original value, i.e. when the network was loaded into memory with AttributeLoad 59.

*Possible error codes:* -10 -30 -31 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.163 SetLinkTime

### Function SetLinkTime(link: integer; time: single): integer;

This function makes it possible to change the time on individual links. This can be used for special links, which doesn't match the road classes: A ferry could be an example, where you want to enter the exact sailing time, because individual ferries don't use the same speed.

Any changes are overwritten after calling function AttributeLoad 59.

Time should be >0.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.164 SetNet

**VCL / CLX / .NET edition only**

### Procedure SetNet(rwnetbase: TRWnetBase);

Links the TRWcalc component to a TRWnetBase component.

*VCL / CLX component:* TRWcalc

**ActiveX edition only**

### Function SetNet(identifier: integer): integer;

Links the RWcalcX component to a RWnetBaseX object.

Possible return codes:
0: OK
-1: Identifier wasn't found
-2: Identifier doesn't point to a RWnetBaseX object

See Identifier 75 function

*ActiveX component:* RWcalcX

## 4.165 SetShortest

### Procedure SetShortest;

This function is a shortcut to defining cost as the shortest route and extra variable as time. It simply sets these 4 parameters this way:

costtime = 0
costdist = 1
extratime = 1
extradist = 0

© 2013 RouteWare / Uffe Kousgaard

See also SetFastest 115

*ActiveX / VCL / CLX component:* RWcalc

## 4.166 SetSpeed

### Procedure SetSpeed(roadclass: smallint; speed: single);

Changes the speed for the present mode and the specified road class (0-31). Default and minimum value is 1 for all road classes.

Speed should be expressed as either km/hour or miles/hour according to Units 126.

Doesn't take effect until AttributeLoad 59 is called.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.167 SharpTurnDrivingDirections

### Property SharpTurnDrivingDirections: integer

See function RouteList 111 for a description.

Default value is 0.

*ActiveX / VCL / CLX component:* RWcalc

## 4.168 ShowProgress

**DLL edition only**

### Procedure ShowProgress(yesno: smallint; exhandle: integer);

ShowProgress(1,exhandle) turns on the progressbar for all functions with progress events 39 defined.

Default value is 0 = no progressbar.

Exhandle is a windows handle to parent the progressbar onto the calling applications window. Specify 0 if unknown.

This only applies to the DLL version.

## 4.169 ShowProgressAvenue

**DLL edition only**

### Function ShowProgressAvenue(yesno: smallint): smallint;

ShowProgressAvenue is a special version for showing progress with ArcView 3.x's built-in progress bar in the same way as function ShowProgress does it.

If the function succeeds, it returns 0, otherwise -1, -2 or -3. If ArcView is installed you should never get an error. Otherwise please contact RouteWare.

© 2013 RouteWare / Uffe Kousgaard

You can't use the built-in progress bars of ShowProgress 118 and those of ArcView at the same time.

## 4.170 SpatialIndexLinksPerCell

### Property SpatialIndexLinksPerCell: integer;

This property can be used to define the spatial index creation (part of function NWcreate 94) to better suit your needs. Optimum value depends on many factors such as typical function calls, type of coordinates 28, geographical distribution of roads etc. so these notes are just guidelines. If you choose the wrong value, performance will suffer, but results will still be correct:

If you have a network with a geographically even distribution of roads all over the onclosing rectangle:
Primarily coordinate2node 64 function calls: 50-200.
Primarily coordinate2location 62 function calls: 10-100.
Mixed use: 50

If the distribution is very uneven (such as Canada):
Primarily coordinate2node 64 function calls: 5.
Primarily coordinate2location 62 function calls: 1.
Mixed use: 2

Default value is 50.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.171 StepsAdd

### Procedure StepsAdd(step: single);

Adds a new value to the list of isochrones generated, when calling function isopoly1new 156 and isopoly2 81. Values <=0 are ignored.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.172 StepsClear

### Procedure StepsClear;

Clears the internal list of steps used for function isopoly1new 156 and isopoly2 81.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.173 SwapOneWay

### Function SwapOneWay: integer;

This function swaps all one-way streets, so the allowed order of travel is reversed. See sample code in Nearest Node 20 for an example on usage.

Returns 0, if no problem occurred.

*Possible error codes:* -10 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.174 TSP2

### Function TSP2(nodenum: integer; TSPtype: smallint; maxseconds: integer): single;

Calculates the optimum order to visit a list of nodes. TSP2 uses a 2-optimum algorithm.

The cost between two nodes are calculated in both directions, but the average is used in the optimization.

Nodenum is the number of nodes in the optimization (2<=nodenum<=NodeListLimit 92).

With 2 GB RAM and with a call to extravarcreate 71, the limit for nodenum is app. 14000. With no call to extravarcreate, the limit is appr. 20000. That many nodes requires several hours of optimization time.

TSPtype = 0 or 10:
A route is calculated, which starts at the first node and ends at the last node in the list. The order of the other nodes are optimized.

TSPtype = 1 or 11:
A round trip is calculated, where the order of all nodes are optimized.

TSPtype = 2 or 12:
A trip is calculated, where only the first node is fixed. The last node on the route will be somewhere far from the first node. This method requires more computing time than type 0 and 1, when ATSP 132=false.

If TSPtype = 10, 11 or 12 straight line distances are used in the optimization. This is a lot faster, but also less accurate.

Maxseconds can be used to limit the total time used for the optimization. Please note that this only applies to the part of the TSP function that actually does the optimization - calculation of the distance matrix is always processed first. Specify 0 to let the procedure run until all possibilities has been tested.

The function returns the minimum cost.

© 2013 RouteWare / Uffe Kousgaard

At the same time the "extra" variable is calculated for the whole route. Get this with function TSP2extra 122.

The new (optimum) order of the nodes can be retrieved with function NodeListGetNewPos 91. Please note this is different from the now obsolete TSP function.

Example input data:
NodeList[1] = 10
NodeList[2] = 17
NodeList[3] = 8
NodeList[4] = 5
NodeList[5] = 12
NodeList[6] = 20

The optimization calculates, that the optimum order is node 8, 17, 5, 12, 20, 10. This is returned in this way:

Example output data:
NodeListGetNewPos[1] = 3
NodeListGetNewPos[2] = 2
NodeListGetNewPos[3] = 4
NodeListGetNewPos[4] = 5
NodeListGetNewPos[5] = 6
NodeListGetNewPos[6] = 1

It is also possible to look it up the other way around, i.e. which position on the ordered list has the Nth node on the list:

NodeListGetOldPos[1] = 6
NodeListGetOldPos[2] = 2
NodeListGetOldPos[3] = 1
NodeListGetOldPos[4] = 3
NodeListGetOldPos[5] = 4
NodeListGetOldPos[6] = 5

The table below shows how well the algorithm performs for 88 standard test datasets from TSPlib. As an example there are 9 datasets with 201-300 nodes. After 1 minute of optimization, the average result is 1.5% worse than true optimum. If the optimization was stopped already after 3 seconds, the average result was 1.6% worse than true optimum. More than 1 minute will not improve it further (unless you have more than 300 nodes). All test runs with <45 nodes was solved to 100% optimality.

| Nodes | Test runs | 3 sec | 1 min | 10 min | 30 min | 4 hours |
|---|---|---|---|---|---|---|
| 1-200 | 40 | 100.7 | | | | |
| 201-300 | 9 | 101.6 | 101.5 | | | |
| 301-1000 | 14 | 103.7 | 102.7 | 102.5 | | |
| 1001-1250 | 5 | 104.9 | 104.1 | 103.5 | 103.4 | |
| 1251-2350 | 14 | 105.9 | 104.8 | 104.2 | 103.8 | 103.6 |
| 2351-6000 | 6 | 106.3 | 105.9 | 105.2 | 104.9 | 104.8 |

If you enter a node number more than once, a short distance will be used for the distance between each occurrence.

© 2013 RouteWare / Uffe Kousgaard

With many nodes, the processing time increases a lot - especially for mode 0, 1 and 2.

Entering node number 0, triggers error -38 and so does nodes in a subnet.

Use OnTSPProgress event for tracking progress.

*Possible error codes:* -10 -30 -38 -40 -41 -43 -50
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.175 TSP2dyn

### Function TSP2dyn(locationnum: integer; TSPtype: smallint; maxseconds: integer): single;

This works exactly the same way as the TSP2 120 function, except it is using the location list functions instead of the node list functions.

## 4.176 TSP2extra

### Function TSP2extra: single;

This returns the extra variable after a call to function TSP2 or TSP2dyn, if extravar 71 was defined in advance.

## 4.177 TurnExport2_BIN

### Function TurnExport2_BIN(filename: string): integer;

This function saves all present turn restrictions to a file on disk, which can later be loaded with function TurnImport2_BIN 123.

If filename is empty the file's name is "turn2_.bin", If filename is "01", the file's name will be "turn2_01.bin" etc. This makes it possible to keep several sets of turn restrictions on disk for the same network.

The file will be saved in the same directory as specified by property Directory 67.

*Possible error codes:* -10 -18 -32 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.178 TurnExport2_GIS

### Function TurnExport2_GIS(filename: string): integer;

This function saves all present turn restrictions to a GIS file on disk, so it is easier to graphically view the turn restrictions.

Supply a full filename, including folder.

© 2013 RouteWare / Uffe Kousgaard

*Possible error codes:* -10 -18 -32 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.179 TurnExport2_TXT

### Function TurnExport2_TXT(filename: string): integer;

This function saves all present turn restrictions to a text file on disk, which can later be loaded with function TurnImport2_TXT 123.

Supply a full filename, including folder.

*Possible error codes:* -10 -18 -32 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.180 TurnImport2_BIN

### Function TurnImport2_BIN(filename: string): integer;

This function loads turn restrictions from a file on disk, which was created by function TurnExport2_BIN 122. If filename is empty the file's name is expected to be "turn2_.bin". If filename is "01", the file's name is expected to be "turn2_01.bin".

The file should reside in the same directory as specified by property Directory 67.

*Possible error codes:* -10 -18 -32 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.181 TurnImport2_TXT

### Function TurnImport2_TXT(filename: string): integer;

This function loads turn restrictions from a text file on disk. Supply a full filename, including folder.

The format is one or more lines, where each line stores one restriction with parameters stored in space seperated format. Different types of restrictions are possible:

0: Simple Turn restriction, 2 external link IDs + 1 cost value
1: Simple Turn restriction 125, 2 link IDs + 1 cost value
2: TurnStandard 126, coordinates for node
3: Mandatory turn, 2 external link IDs
4: Mandatory turn 124, 2 link IDs
5: Complex Turn restriction, >2 external link IDs + 1 cost value
6: Complex Turn restriction 125, >2 link IDs + 1 cost value

File example:

© 2013 RouteWare / Uffe Kousgaard

```
// Comment
0 A4003234 A4003127 -1
1 456 230 -1
2 -77.024098 38.902711
3 A4003234 A4003127
4 456 230
5 A4003279 A4003234 A4003127 -1
6 89 456 230 -1
```

Lines starting with // are ignored as comments.

The free ITN converter will create turn restriction files in this format.

If you use turn restrictions with external ID's (type 0, 3 and 5), make sure you have called ExternIDOpen 70 first or you will get an error code returned.

Type 0, 2, 3 and 4 gets translated into one or more type 1 during import and type 5 gets translated into type 6 during import.

*Possible error codes:* -10 -18 -32 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.182 TurnMandatory

### Function TurnMandatory(link1,link2: integer): integer;

Defines that turns from link1 is only allowed if the next turn is link2.

Internally this is translated into a number of turn restrictions. These are only applied at the end of link1, where it is actually possible to use link2.

If link1 and link2 are parallel links, you will get a -30 error.

*Possible error codes:* -10 -18 -30
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.183 TurnMode

### Property TurnMode;

Defines if routing supports turn restrictions or not.

False: Standard routing (default value)
True: With turn restrictions

You can only change turnmode if the network isn't loaded. Turnmode can not be changed in free version.

*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.184 TurnReset

### Procedure TurnReset;

Clears the list of turn restrictions.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.185 TurnRestriction

### Function TurnRestriction(link1,link2: integer; cost: single): integer;

Defines a restriction on turns from link1 to link2.

cost<0: Turn prohibited
cost=0: Remove turn restriction
cost>0: Additional cost related to the turn (=delay).

If link1=link2 the restriction (a U-turn) is skipped. See here 126 how to apply U-turn restrictions. It is not possible to have delays for U-turns, they can only be either allowed or banned.

If link1 and link2 are parallel links, a turn restriction is added at both nodes. Prevent this by breaking up one of the links.

*Possible error codes:* -10 -18 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.186 TurnRestrictionComplex

### Function TurnRestrictionComplex (link1,link2,link3,link4,link5,link6: integer; cost: single): integer;

Defines a restriction on turns from a sequence of links. Up to 6 links can be defined. If you have <6 links, just set the remaining values to 0. If you have more than 6 links, it is required to use the TurnImport2_TXT for loading the restrictions.

Currently there is no actual support for complex restrictions in the calculations, but it will be added later and they can already now be imported and exported through the functions for this.

cost<0: Turn prohibited
cost=0: Remove turn restriction
cost>0: Additional cost related to the turn (=delay).

*Possible error codes:* -10 -18 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.187 TurnStandard

### Function TurnStandard(node: integer): integer;

Adds turn restriction on standard 4-road intersections (valency=4), which means no turns are allowed - only driving straight through.

From version 2.21 and on, this can be applied to any node with even valency >= 4.

*Possible error codes:* -10 -18 -30 -43 -46
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 4.188 Units

### Property Units;

Units can be either Km (default) or Miles. Distances are returned according to this. Units has no influence on NWcreate 94.

Units should be defined before you call function NWload. Changing it later on may lead to undesired results.

Speed as defined in SetSpeed 118 should be km/h or miles/h according to this.

Time is always measured in minutes.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.189 UTurnAllowed

### Property UTurnAllowed;

Defines if U-turns are allowed when Turnmode 124=true.

False: All U-turns are banned (default).
True: All U-turns are allowed, unless banned through attribute 27 settings.

*ActiveX / VCL / CLX component:* RWcalc

## 4.190 Valency

### Function Valency(node: integer): integer;

Returns the valency of the node. Valency is the number of links connected to a node.

Valency is a number from 1 to 1900, which is the highest possible valency in RW Net.

*Possible error codes:* -10 -30
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 4.191 Version

### Property Version;

Returns set constant indicating the version, e.g. "Standard_v200".

In DLL an integer is returned: 0=Free, 1=Standard and 2=Pro.

*ActiveX / VCL / CLX component:* RWnetBase

## 4.192 ViaListClear

### Procedure ViaListClear;

Clears the vialist.

*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 4.193 ViaListSet

### Function ViaListSet(index: integer; name: string; time: double): integer;

The Via List can be used to define a list of places to visit on a route. In particular this includes a name and a time (in minutes) to spend on each place. The function should be used together with function RouteList 111. Index should be >= 1.

To define the geographical position of the places, use either NodeListSet 92, LocationListSet 88 or PositionListSet 99.

An example with 4 places:

```
ViaListSet(1,"Start",0)
ViaListSet(2,"Customer 1",5)
ViaListSet(3,"Customer 2",3)
ViaListSet(4,"Stop",0)
```

The list will automatically grow as you enter more entries. Default value is an empty string and 0 minutes. See also ViaListClear 127.

The function can also be used with isochrone generation, see IsoCostOffSet 79.

*Possible error codes:* -10 -30
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 4.194 ZLevelFile

### Property ZLevelFile: string

If this property isn't blank it specifies the name of a file with Z-level information from TomTom files. Z-level is an integer from -9 to 9, which specifies the level of streets: One number for the start of the line and one number for the end of the line.

The file should be a standard text file with 2 numbers on every line: Z-level for start and end of line. The numbers should be separated by a white space. There should be the same number of lines as records in the base file.

When this property points to a valid filename, the information is used as part of function NWcreate 94 and coordinates are adjusted slightly (10 cm = 4 in.) to prevent end-nodes at different Z-levels to have the same coordinates. Only if Z-level<>0, the modification is applied.

Filename is without path, but the file should be in the directory specified by property Directory 67.

*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

# Part V

# Reference: Pro only

# 5 Reference: Pro only

RW Net Pro includes these functions not found in RW Net Standard:

- Ability to work with larger networks
- Assignment 131 (traffic assignment)
- ATSP 132 (asymmetric travelling salesman algorithm)
- CPP 132 + EulerRoute 139 (chinese postman problem)
- District 136 (school districting etc.)
- Encryption 138 of binary files
- Hierarchical 36 routing
- Node2Link 145
- NWexport 146

These are mostly functions for finding problems in your network:

- Cul-De-Sac 135
- FindCloseNodes 139
- Join links 143
- NodeLinkCheck 146
- Overpasses 147
- Parallel links 147
- RoadNameGroupBy 148
- RoadNameTest 148
- Subnet 150
- SubnetEx 150
- UnUsedLinks 151

## 5.1 Assignment

### Function Assignment(filename: string; Atype,Grouped: smallint): integer;

Assigns traffic to a network based on a list with traffic volumes between pairs of coordinates or node numbers.

Filename holds the name of the file with traffic to be assigned. This is the full filename and doesn't rely on property "Directory".

If Atype=0 each line in the file holds pairs of nodes and the traffic volume:
<node1 node2 volume>

If Atype=1 each line in the file holds pairs of coordinates and the traffic volume:
<x1 y1 x2 y2 volume>

If Grouped=1, it is assumed that multiple occurrences of node1 or (x1,y1) is grouped together.
If Grouped=0, it is assumed they are in random order.

The value of Grouped doesn't matter, but performance suffer if the wrong value is used as input.

© 2013 RouteWare / Uffe Kousgaard

If a traffic volume can not be assigned because the corresponding nodes are not connected, the function stops and returns a positive number which contains the line number with the problem.

The result can be saved with ResultSaveLink.

*Possible error codes:* -10 -40 -41 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.2 ATSP

### Property ATSP: boolean;

ATSP is short for Asymmetric Travelling Salesman Problem.

Setting this property to true, makes the TSP2 120 and TSP2dyn 122 functions use another algorithm, that is better suited for street networks with many one-way restrictions (i.e. asymmetric distance matrix). In such instances the improvement of the solution can be as much as 4-8% (depends a lot on the actual data) compared to the normal 2-optimal algorithm. In situations with symmetric distance matrices, the solution is app. of the same quality as the normal algorithm.

Calculation time is generally the same for smaller problems (0-50 nodes), but it is faster than the normal algorithm for larger problems (significant improvements can be achieved).

The algorithm uses random permutations (known as simulated annealing), so you can not always get the same result back between different runs on the same input data.

In the free/standard version ATSP is always false.

*ActiveX / VCL / CLX component:* RWcalc

## 5.3 CPP

### Function CPP(mode: smallint; factor: double; startnode,endnode: integer; filename,attributefile: string): integer;

The Chinese Postman Problem (CPP) deals with visiting all links in a street network such as is the case of many real-world tasks: Snow-ploughing, mail delivery, checking street lights etc. This function solves the CPP, by adding additional links to the network and thereby turning it into an Eulerian Graph.

Many different variations of CPP exist and mode is a constant, which describes the kind of problem to be solved:

0: All links will have to be visited once, no links has one-way restrictions and travel between link-ends is not limited to the links.
Example: Inspection of a network by helicopter

1: All links will have to be visited once, no links has one-way restrictions.

© 2013 RouteWare / Uffe Kousgaard

Example: Inspection of a street network (walking).

2: All two-way links will have to be travelled in both directions, one-way links only once.
Example: Snow-ploughing

3: All links will have to be travelled once, one-way streets in the correct direction.
Example: Inspection of a street network (by car).

So far mode 0, 1 and 2 has been implemented in RW Net Pro.

The whole idea behind solving the CPP, is to minimize the amount of additional travel, that is travelling on links, which has already been visited.

The factor parameter (between 0.1 and 1) defines how many possible combinations of additional travel is included in the optimization. For small datasets, just use 1. For larger datasets reduce to 0.25, to speed up the calculation time. If factor becomes too low, you will get error -54.

It is possible to define a startnode and endnode if you don't want a round-trip as result. 2 different and valid node numbers will force the resulting dataset to have un-even degree for the 2 specified nodes. Otherwise just specify 0 for both nodes.

It is possible to define which links **has** to be visited and which are optional, so they are only used if it reduces the overall length of the Eulerian Graph. To do this call function SetLinkResult 116 with 0 or 1 for all links:

0: Optional
1: Has to be visited

If optional links are chosen, so that the graph is disconnected without these, you risk getting a disconnected output graph as a result, which is not an Eulerian Graph. No error is returned in that case.

Output from the function is a new dataset with filename as defined in the function call. The new dataset contains all the links of the original dataset plus some new ones, which may be duplicates of the old ones. The new dataset will contain a linkID, a reference to the original linkID from the input dataset, a logical field which describes, if it is a direct duplicate and one-way information. By using the linkID field, it is possible to link the new dataset with further attribute fields from the original dataset. This could be road class, street name etc.

It is optionally possible to store one-way attributes of the new dataset in a textfile, ready for being read by function AttributeCreate 58. Use parameter attributefile for this. If this parameter is an empty string, no attribute file is created. This is only really relevant for mode 2.

A progress event is available (CPPprogress) that also allows you to break off the processing before finalization, if the processing time seems to be too long. Street networks with 1000-2000 links is considered a large dataset for this procedure and may result in computing times up to ½ hour. Exact time depends on many factors such as network structure, *mode* and *factor*, so it isn't possible to give precise estimates, but mode 0 is the slowest, followed by mode 1 and then mode 2.

© 2013 RouteWare / Uffe Kousgaard

Format of the new dataset is as defined in property GISformat 74.

The return value of the function is the length of additional travel (mode 0) or cost of additional travel (other modes).

Once a new dataset has been generated, you should call function EulerRoute 139 to generate the actual route.

*Possible error codes:* -10 -18 -30 -32 -40 -41 -50 -52 -53 -54
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

This sample code assumes a SHP file called "link" and in the case of a mode 2 calculation, you also need a text file with the attributes in.

```
mode = 1
NWCreateSHP("link",false,"")
if mode=2 then
  AttributeCreate("attribute.txt")
end if
NWload()
if mode=2 then
  RWnetBase1.AttributeLoad(true);
end if

All links in the shp file should be included in the route
for i = 1 to rwnetbase1.linkmax()
  SetLinkResult(i,1)
next i

startx = 1010
starty = 23009
stopx = 562
stopy = 23409
if different_nodes=true then
  The route should start and end in these nodes
  Coordinate2Node(startx,starty,startnode,dist)
  Coordinate2Node(stopx,stopy,endnode,dist)
else
  Any nodes will do
  startnode = 0
  endnode = 0
end if

Main processing
if mode=2 then
  AddCost = CPP(mode,1,startnode,endnode,"cpp_out","attribute_cpp.txt")
else
  AddCost = CPP(mode,1,startnode,endnode,"cpp_out","")
end if
NWUnLoad()

if AddCost>0 then
  print "Additional cost: "+str$(result)

  process output
  NWCreateSHP("cpp_out",true,"")
```

© 2013 RouteWare / Uffe Kousgaard

```
  NWload()
  if mode=2 then
    AttributeCreate("attribute_cpp.txt")
    AttributeLoad(true)
  end if

  find new startnode based on same coordinates as before
  Coordinate2Node(startx,starty,startnode,dist)

  Create euler route - final result
  length = EulerRoute(startnode)

  if length-1<>LinkMax() then
    print "Not a fully covering route"
  end if

  Generate a new table with route as result
  RouteFileCreate("route",2)
  RouteFileFieldAdd("linkID",field_integer,0,0)
  RouteFileFieldAdd("Order",field_integer,0,0)
  dist = 0
  for i = 1 to length-1
    t = RouteGetLink(i)
    RouteFileRecordAdd(1,str$(t)+" "+str$(i))
    if t>0 then
      RouteFileRecordAddSection(t,0,1)
    else
      RouteFileRecordAddSection(-t,1,0)
    end if
    dist = dist+GetLinkDist(abs(t))
  next
  RouteFileClose()
  print "Route : "+str$(dist)+" km"
  NWUnLoad()
else
  print "Error: "+str$(AddCost)
end if
```

## 5.4 CulDeSac

### Function CulDeSac(Ctype: smallint): integer;

Identifies cul-de-sac or blind alleys, which are defined like this:

Position yourself in a node and drive in one direction using link A. If you can't get back to the same node without using the same link A again, it is a cul-de-sac (type 1).

Loop links are type 2.

If you however can come back without making a U-turn, it is a bridge or isthmus in graph theory terms: If the link was missing, the network would be split into 2 components or subnets (type 3).

If all type 1, 2 and 3 links are removed, the rest of the links are separated into a number of subnets. Type 4 is now the subnets identified in the same way as function SubNet does it, that is the smallest networks are defined as subnets, while the rest is

© 2013 RouteWare / Uffe Kousgaard

considered the main part of the network. Number of links in the subnets defines "size". But note that subnets are not defined the same way as returned in function SubNet.

0 = normal roads (the rest)
1 = cul-de-sac
2 = loops
3 = bridge
4 = subnets

Ctype=1: Type 0, 1 and 2 are identified.
Ctype=2: Type 0, 1, 2 and 3 are identified.
Ctype=3: Type 0, 1, 2, 3 and 4 are identified.

The result can be saved with ResultSaveLink 101 or ResultGetLink 101.

Note that Ctype 2 and 3 take a lot more time to identify and you will have to call AttributeLoad 59 after using this function as all information about oneway streets are reset.

*Possible error codes:* -10 -40 -41 -42 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.5 District

### Function District(method: integer; centerfile, customerfile, outputfile: string): integer;

This function solves the problem of assigning students to schools, where each school has a maximum capacity. The criteria for assigning students is to minimize the total cost (normally distance) for all students. The function can be used for other problems too and therefore schools are also referred to as centers, while the students are referred to as customers below.

Centerfile is the filename of a text-file, which contains the information for the centers. Include one line for each center with this space-delimited information: X-coordinate, Y-coordinate and capacity.

Customerfile is the filename of a text-file, which contains the information for the customers. Include one line for each customer with this space-delimited information: X-coordinate and Y-coordinate.

There can be any number of centers (>= 1) and customers (>= 1). If the total capacity of the centers is smaller than the number of customers, only some of the customers will be assigned to a center.

The outputfile is the name of a text-file with the results from the calculations. It looks like this:

ID Center Priority Cost
1 4 1  0.657
2 0 0  0.000

© 2013 RouteWare / Uffe Kousgaard

3 0 0  0.000
4 0 0  0.000
5 3 1  1.331
6 0 0  0.000
7 0 0  0.000
8 3 1  1.125
9 3 1  1.176
10 3 1  1.091

ID refers to the number of the customer in the customerfile (line number).
Center is the number of the center, which the customer has been assigned to. Here both customer 2, 3, 4, 6 and 7 have not been assigned.
Priority is 1, of the customer has been assigned to the nearest center, 2 if it was the 2nd nearest etc.
Cost is the actual cost of getting there.

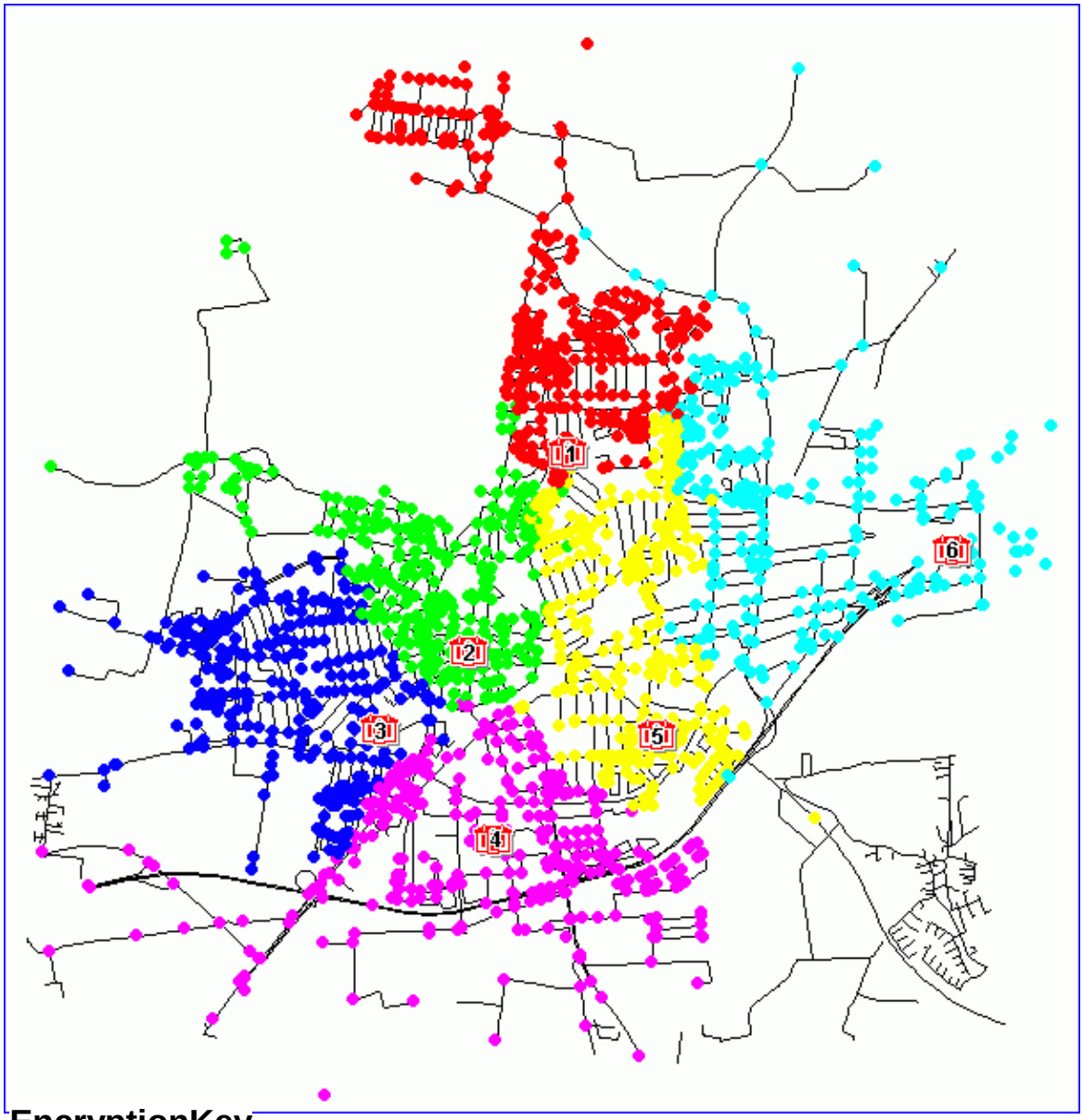
The function returns 0, if no errors happened.

The Method parameter is not being used at the moment, but will later be used for adding more district optimization methods.

*Possible error codes:* -10 -32 -40 -41 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

This map shows an example of 6 schools and assignment of students. It can easily be seen, that school 1 has too little capacity since many of the nearest students has been assigned to other schools with sufficient capacity (both schools 2, 5 and 6):

© 2013 RouteWare / Uffe Kousgaard




## 5.6 EncryptionKey

### Property EncryptionKey: integer;

By setting this property the binary network files gets encrypted during creation. Make sure to use same key both when calling function NWcreate 94 and NWload 97 (and AttributeCreate 58 / AttributeLoad 59, LimitCreate 84 / LimitLoad 86). If different keys are used for creation and loading, the loading will fail.

Only some of the binary files are encrypted - see list of files 29 for details.

Having encryption enabled makes network creation and loading slower. This is an example from a large dataset (6 million records):

| Seconds | NWcreateTAB | NWload | NWload w/ caching of coord3.bin |
|---|---|---|---|
| Normal | 1298 | 22 | 38 |

© 2013 RouteWare / Uffe Kousgaard

| | | | |
|---|---|---|---|
| With encryption | 1536 | 89 | 130 |

If you choose to cache 61 the content of coord3.bin file in memory, it will get unencrypted at load time. Otherwise it will get unencrypted piece by piece when needed: This will slow down some functions relying on it. See chapter file structure 29 for a list of functions relying in coord3.bin.

In Free and Standard versions, you can not set the property and can therefore not open encrypted networks or create them.

Default value is 0 (not encrypted).

*Versions:* Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 5.7 EulerRoute

### Function EulerRoute(startnode: integer): integer;

This function works in conjunction with function CPP 132. Startnode should be any valid node number. If function CPP was called with different start- and endnode, make sure both startnode parameters point to the same set of coordinates on the network, which however will not have the same node number (!).

The result from the function is the number of nodes on the generated route and the generated route can be checked by the usual route functions: RouteGetLink 110 etc.

If EulerRoute()-1<>LinkMax 87, the calculated route will not cover the whole graph. This is an error situation, if the network is believed to be Eulerian.

Please see sample code supplied with function CPP 132.

The function will not respect turn restrictions.

*Possible error codes:* -10 -30
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.8 FindCloseNodes

### Function FindCloseNodes(snap: single): integer;

Identifies all nodes, which are positioned closer to each other than specified in snap (km or miles according to unit 126). The result is written to GIS file "closenodes". The format of the GIS file is determined by property GISformat 74.

Close nodes are often an indication of problems in the network, e.g. missing "snap".

The function may identify some of the same problems as the other network checking routines (NodeLinkCheck 146 in particular) and doing a simple check for very short links is always a good idea, before running the routine, since these can trigger many "false" problems.

© 2013 RouteWare / Uffe Kousgaard

The generated file will contain the distance between the nodes. If the 2 nodes are connected directly by a link, they are not part of the output.

*Possible error codes:* -10 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 5.9 FindRoundAbout

### Function FindRoundAbout(maxlink: smallint; maxradius: single): integer;

Locates links, which are roundabouts.

By definition roundabouts:
1) Need to have at least 3 links,
2) Should form a loop,
3) All nodes in the roundabout should have a valency of 2 or 3,
4) At least 3 of the nodes should have a valency of 3.

Maxlink denotes the maximum number of links in a roundabout. Max has to be >2 and <100.

Maxradius is the maximum radius of the roundabout.

The function locates roundabouts using two methods:
First it will look for links with one-way streets. If a loop can be found, where a sequence of links all are one-way streets in the same direction, it will be considered to be a roundabout.

Next it will search for sequences of links, where various geometric properties will also have to be in order to be accepted as a roundabout.

The method isn't 100% bullet proof, so a manual inspection of the result is needed. Usually most roundabouts are located, but also some false ones.

Networks with latitude/longitude coordinates are not supported by this function, however you will not be stopped if you try.

The function returns the number of round-abouts.

The result can be saved with ResultSaveLink / ResultGetLink and is defined as this:

0: Not roundabout
1: Links belonging to first roundabout
2: Links belonging to next roundabout etc.

Use OnCalcProgress event for tracking progress.

*Possible error codes:* -10 -16 -30 -40 -41 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 5.10 Hierarchy

### Property Hierarchy: boolean;

You can only change this property, if a hierarchy file has been loaded 143.
See also Hierarchical routing 36

*ActiveX / VCL / CLX component:* RWcalc

## 5.11 HierarchyCheck1

### Function HierarchyCheck1(filename: string): integer;

This function will verify the integrity of the hierarchical attribute. The rule is that each subset of the full street network based on hierarchy levels 1, 1-2, 1-2-3 and 1-2-3-4 need to be a fully connected street network. Level 1-2-3-4-5 is the full database and can be checked in a similar way by function subnet 150 etc.

Nodes with valency 1 are only allowed in each of the 4 sub-networks, if they are the ones at the border of network (neighbouring countries for instance).

The output is stored as a GIS point theme with nodes, that show where possible problems are.

Filename should include a fully qualified path.

This function and HierarchyCheck2 141 will both find many of the same problems in data, but there are also situations which will only be discovered by one of the methods. We suggest that you first run this function, correct any problems and then run HierarchyCheck2 to see if any problems are left.

See also Hierarchical routing 36.

*Possible error codes:* -10 -32 -40 -63
*ActiveX / VCL / CLX component:* RWnetBase

## 5.12 HierarchyCheck2

### Function HierarchyCheck2(filename: string): integer;

This function will verify the integrity of the hierarchical attribute. The rule is that each subset of the full street network based on hierarchy levels 1, 1-2, 1-2-3 and 1-2-3-4 need to be a fully connected street network. Level 1-2-3-4-5 is the full database and can be checked in a similar way by function subnet 150 etc.

The output is stored as a GIS polyline theme, that show where possible problems are.

Filename should include a fully qualified path.

See also Hierarchical routing 36 and HierarchyCheck1 141

*Possible error codes:* -10 -32 -40 -63
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 5.13 HierarchyCreate2

### Function HierarchyCreate2(filename,fieldname: string; fieldindex: integer; netbclass: boolean): integer;

From the specified filename a file called hierarchy.bin is created, which holds all attributes regarding the links.

The file should be of type DBF (version 3) or DAT file (part of a TAB file).

In the case of a DAT file, the corresponding TAB file needs to be present.
If you supply fieldindex=0, the fieldname will be used. If you supply a fieldindex (1..) it will be used for reading from the file.
If the field, you are pointing at, is a floating point field, it will be rounded.

The values in the field should be in the range 1 to 5. Values outside of this range will be set to 5.

If netbclass=true, it is assumed you are reading from the TomTom, netbclass field (range 0..6) and it will on the fly be recoded into the 1..5 range similar to the obsolete net1class field (range 0..4).

If you are using Navteq data, the field func_class can be used as it is.

Filename should include a fully qualified path.

Hierarchy.bin simply holds the values from the input file and so it is possible to write directly to this file if needed. It consists of 8-bit unsigned integers. You may even create the file from scratch yourself, if that suits you better, but you miss being able to encrypt the file.

Use OnAttributeCreateProgress event for tracking progress.

**Examples:**
HierarchyCreate2("network.dbf","func_class",0,false);

HierarchyCreate2("network.dbf","netbclass",0,true);

See also Hierarchical routing 36

*Possible error codes:* -10 -31 -40
*ActiveX / VCL / CLX component:* RWnetBase

## 5.14 HierarchyLevelSet

### Procedure HierarchyLevelSet(h2, h3, h4, h5: double);

Sets the 4 hierarchy parameters for use in hierarchical routing. Values should be expressed in km or miles, depending upon property units 126.

Input requirement: h2 >= h3 >= h4 >= h5 >= 0.
By default all parameters are set to infinite, meaning no hierarchy is applied.

© 2013 RouteWare / Uffe Kousgaard

We have executed tests with TomTom (*netbclass* field) and Navteq (*func_class* field) databases and recommend these values:

| | Km | Miles |
|---|---|---|
| TomTom | 130, 120, 100, 22 | 81, 75, 62, 14 |
| Navteq | 145, 90, 40, 7 | 90, 55, 25, 4.4 |

Tests were executed on UK data with a large number of random routes. Compared to not using a hierarchy, calculations were 6 times faster with TomTom data (0.3 secs per route) and 11 times faster with Navteq data (0.1 sec per route). Navteq has better hierarchy attributes and a little less details in the network, hence the differences.

For short routes (<50 km) there is only little difference between using a hierarchy or not, while calculation of longer routes (>400 km) in the UK may be as much as 20-40 times faster (Navteq) and 6-30 times faster (TomTom).

See also Hierarchical routing 36

*ActiveX / VCL / CLX component:* RWcalc

## 5.15 HierarchyLoad

### Function HierarchyLoad: integer;

Loads an existing hierarchy file ("hierarchy.bin") into memory.

Use OnAttributeLoadProgress event for tracking load progress.

See also Hierarchical routing 36

*Possible error codes:* -10 -31 -40
*ActiveX / VCL / CLX component:* RWnetBase

## 5.16 Join

### Function Join(maxlength: single; filename: string): integer;

Join locates which links in a network can be joined together. If a node has only two links connected to it, that node may be superfluous for some purposes (aka. a pseudo-node). For other purposes it is needed, e.g. if you want to calculate the distance from that node to another or if the name of street or any other attribute change at that position.

Route calculations on a network with many pseudo nodes is slower and requires more RAM than the same one without pseudo nodes. This is the main reason for removing and compacting a network.

If a network has many very short links, which you would like to join, you can specify that links should be joined until they reach a length of maxlength. If maxlength=0, there is no upper limit and all links between nodes with more or less than 2 links, will be connected.

© 2013 RouteWare / Uffe Kousgaard

If a filename is specified (including full path), it should for each link contain a line with the street name. This can be used to stop the joins every time the street name changes on the links. If the file isn't found, the calculation is done as if the filename is "". The file can of course hold any string value.

The result can be saved using ResultSaveLink 101 / ResultGetLink 101. All links with the same value>0 should then be joined/combined in a GIS-program. Now you can create a new topological network, which is smaller and faster.

Use OnJoinProgress event for tracking progress.

See also Join2 144.

*Possible error codes:* -10 -40 -41 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.17 Join2

### Function Join2(maxlength: single; ID: integer): integer;

This is the same function as Join 143, expect it uses one of the already loaded roadname databases 103 (ID parameter) for reference instead of an external text file. This is more memory efficient and faster than function Join.

*Possible error codes:* -10 -30 -40 -41 -43 -49
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.18 NetworkCenter

### Function NetworkCenter(NodeNumber, MaxIterations: Integer; Eps, power: Double; var TotalCost: Double): integer;

This function solves the minimax problem for the street network. This means locate a number of facilities in a network, so the distance from any node to the nearest facility is minimized. The method uses an allocate-locate strategy.

Input parameters are *NodeNumber*, which tells how many facilities you want.

*Maxiterations* controls the granularity of a grid used for testing additional nodes to prevent getting stuck in local optima. We recommend 0 to 3. 3 gives for instance 3 x 3 = 9 nodes. With higher values you get much longer calculation times, but 2 is generally sufficient.

*Eps* controls the relative difference in *TotalCost* value between master iterations, that make the calculations stop. We recommend 0.01 (i.e 1%). Do not use 0 or you risk an endless loop.

Power controls the kind of target:
1) With power=100 it uses the classical minimax strategy, i.e. all parts of the network are treated equally important.
2) With power=1 it minimizes the cost to each link from the center multiplied by the

© 2013 RouteWare / Uffe Kousgaard

length of the link. Here the length of the link serves as a weight or proxy for number of inhabitants etc.
3) With power>1 the cost above is raised to the power of the value. Typical values will be from 1 to 2.

You can fill in the nodelist in advance with the location of existing facilities, since last step in the calculations is minimizing the total distance between existing locations and new centers, by sorting the new centers so they match with the previous locations.

Result is available through function NodeListGet 91.

See also NetworkCenter2 145.

*Possible error codes:* -10 -30 -40 -41
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.19 NetworkCenter2

### Function NetworkCenter2(NodeNumber, MaxIterations: Integer; Eps, power, cutoff: Double; var TotalCost: Double): integer;

Same function as NetworkCenter 144, except weight for each link is supplied by calling SetLinkResult 116 and an additional cutoff parameter is available.

Cutoff is used when power<>100: If cost < cutoff, the link is not included in calculation of totalcost. If cost > cutoff, (cost-cutoff) is used in the calculation of totalcost.

This can be used for location fire stations for instance. If cutoff is 10 minutes, the optimization process will focus on minimizing drive times beyond 10 minutes. Anything below 10 minutes will be considered within the limit.

*Possible error codes:* -10 -30 -40 -41 -64
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.20 Node2Link

### Function Node2Link(node, linkindex: integer): integer;

Returns the ID of the link starting at node. Linkindex should be <= the valency of node.

This function is typically used together with Link2FromNode 86, Link2ToNode 86 and Valency 126 to perform user specific network topological analysis.

*Possible error codes:* -10 -30 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 5.21 NodeLinkCheck

### Function NodeLinkCheck(filename: string; buffer: double): integer;

This function checks the validity of the network by creating a small buffer around all network nodes and checking if all links within the buffer is connected to the node. Any place, where this is not the situation, is stored in a GIS file for later inspection.

Buffer should be in the same unit as units 126. Use a rather small value for buffer such as 0.001 for 1 meter or perhaps even smaller.

Supply a full filename including path for output, but excluding extension. The format of the GIS file is determined by property GISformat 74.

The output file contains 2 integer fields with this information:
Node: ID of the node
Link: ID of non-connected link

The function returns the number of problems found or an error code.

The function may identify some of the same problems as the other network checking routines (FindCloseNodes 139 in particular) and doing a simple check for very short links is always a good idea, before running the routine, since these can trigger many "false" problems.

Use the OnAnalyzeProgress event for tracking progress.

Performance example: Checking a network with 590,000 links took 7 minutes on a AMD64 based computer. If possible make sure coord3cache 61 = true as in this example.

*Possible error codes:* -10 -30 -44 -46 -50
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 5.22 NWexport

### Function NWexport(filename: string): integer;

Exports the currently loaded network into a new GIS file in the format specified by property GISformat 74. The generated file contains attribute information about the original record ID.

Deleted records are skipped in the output and any formatting in the original MIF/TAB/CGF files is lost. Objects, which are not single-sectioned (poly)lines, are skipped too. If any objects has a length of zero (i.e. a polyline with the same 2 vertices), they are also skipped, since they are not part of the network. The generated GIS file will also reflect any ZLevelFile used when creating the network.

Use OnNWexportProgress event for tracking progress.

*Possible error codes:* -10 -32 -40 -46

© 2013 RouteWare / Uffe Kousgaard

*Versions:* Pro (not trial licenses)
*ActiveX / VCL / CLX component:* RWnetBase

## 5.23 Overpasses

### Function Overpasses(filename: string): integer;

Identifies all pairs of links, that intersect at other points than the first / last vertex. Self-intersections are not tracked (see ExtendedDataCheck 68 instead) and only 1 overpass is tracked per pair of links. Overpasses may either be a bridge or a topological error in the network.

Supply a full filename including path for output, but excluding extension. The format of the GIS file is determined by property GISformat 74.

The output file contains 3 integer fields with this information:
Code:
1: Overpass
2: Duplicate vertex in object
3: Exact same 2 objects
4: Exact same 2 objects, reverse direction
Link1: ID of first link
Link2: ID of second link

If code is 2, link2 is always 0.

The function returns the number of overpasses found or an error code.

Use the OnAnalyzeProgress event for tracking progress.

Performance example: Locating 3,000 instances in a network with 590,000 links took 50 secs on a AMD64 based computer. If possible make sure coord3cache 61 = true as in this example.

*Possible error codes:* -10 -32 -40 -44 -46 -50
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 5.24 ParallelLinks

### Function ParallelLinks: integer;

Identifies group of links, which start and end at the same two nodes. These might give problems in some networking algorithms ("emme/2" for instance).

RW Net has no problem with parallel links, unless you want to apply a turn restrictions from one parallel link to another and only want it at one of the 2 nodes they have in common.

The function returns 0, if no errors occured.

Return values from the computations:
0: normal links

© 2013 RouteWare / Uffe Kousgaard

1: Single link, which start and end at the same node (loop).
>=10: Shortest link in group
<=-10: Other links in group

Three parallel links would have the value 10 (the shortest) and -10 for the other two. The next group of parallel links has value 11 / -11 and then 12 / -12 etc.

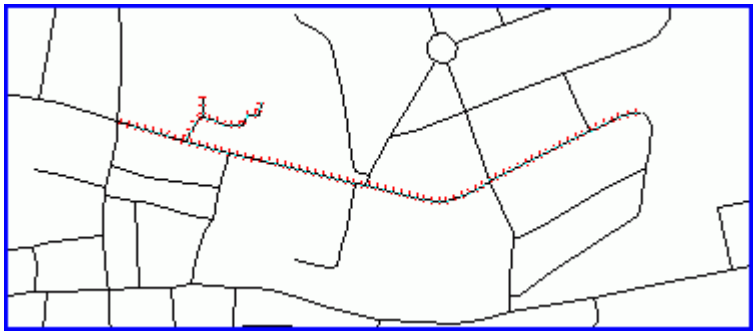
The result can be saved with ResultSaveLink 101 / ResultGetLink 101.

*Possible error codes:* -10 -40 -41 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.25 RoadNameGroupBy

### Function RoadNameGroupBy(RoadNameID: integer): integer;

This function detects groups of streets with the same road name, that are also connected to each other. This can be used for later joining of all records within the same group, so a specific road can be highlighted on a map with a single click:



RoadnameID defines the database with road names to use with the function.

The "group by" value for each link in the network can be extracted with function ResultGetLink 101 / ResultSaveLink 101.

The total number of groups are returned by the function.

*Possible error codes:* -10 -49
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.26 RoadNameTest

### Function RoadNameTest(filename: string; RoadNameID, NumOfRoutes: integer): integer;

This function tests the roadname as part of the driving directions. The theory is, if a road name occur more than once on a route, it may be an error and the links between the 2 occurences might have the wrong name in the database. An example:

Link 1, 2, 3: Main Street

© 2013 RouteWare / Uffe Kousgaard

Link 4, 5: Old Road
Link 6, 7: Main Street

Here link 4 and 5 should probably have had the name Main Street as well. The function will report such instances and then leave it to the user, if any edits should be performed.

RoadnameID defines the database with road names to use with the function and NumOfRoutes defines how many random routes to calculate as part of the test.

Filename specifies the name of the output table. Format is as defined by property GISformat 74.

The fields in the generated GIS file are linkID, count of links that should be changed in the same way (2 in the example above), present roadname and suggested roadname. Generally, a low number of links to be changed, indicates a higher likelihood, that it is a required change. Using a filter of count<20 is a good idea, before viewing the output. You may also ignore roundabout links.

The value of count can also be extracted with function ResultGetLink 101.

*Possible error codes:* -10 -30 -32 -40 -41 -43 -49
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.27 SetApproach

### Procedure SetApproach(approach1, approach2: integer; viapoints: boolean);

This procedure works in connection with function RouteList 111 and makes it possible to define how the generated route should start and end, if it uses dynamic segmentation. For node based routes, it has no influence.

If approach1 = 512 the route will start in the direction of digitization for the first link. If the value is 1024 it will be opposite. If the value is 0 (default) it will be whatever is the shortest / fastest.

In the same way approach2 manages how the route will end.

If parameter viapoints is true, all viapoints will be approached from one side and the route will then continue, so no U-turn is required.

Depending on oneway restrictions in the network, one or more of these rules may be broken in the output.

*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

## 5.28 SubNet

### Function SubNet: integer;

Calculates which part of a network is a subnet. A subnet is defined as a part of the network, which isn't connected to the rest of the network. This is done as if all streets can be travelled in both directions, so one-way restrictions may in fact make even more streets in-accessible. See function SubNetEx 150 on how to detect such situations.

The main net is defined as the part of the network, where the node with highest valency is found (node number 1).

The result for each link can be saved with ResultSaveLink 101 or ResultGetLink 101:

0 = main net
1,2,3... = sub nets

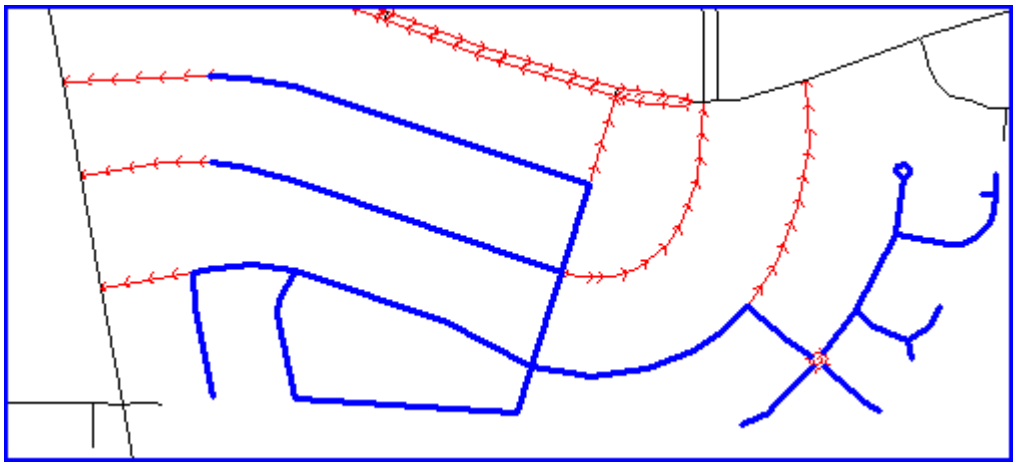
Returns the number of subnets.

*Possible error codes:* -10 -16 -40 -41 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.29 SubNetEx

### Function SubNetEx: integer;

Detects if a route between any 2 nodes can only be found when going in one of the directions. The links with the problematic oneway restrictions are identified. In the example below it is only possible to drive *away* from the blue area, since all oneway restrictions point in the same direction. The function would identify the 6 links with oneway restrictions. These are not fake data, but from a commercial database!



The result for each link can be saved with ResultSaveLink 101 or ResultGetLink 101:

0 = No problem
1 = Problem

© 2013 RouteWare / Uffe Kousgaard

Returns the number of problematic links.

*Possible error codes:* -10 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

## 5.30 UnUsedLinks

### Function UnUsedLinks: integer

This function returns the number of unused links in the network. An unused link is one which isn't part of any route at all (with the exception of dynamically segmented routes). This is determined using the current cost function.

The result for each link can be saved with ResultSaveLink 101 or ResultGetLink 101:

0 = Normal link
1 = Unused
2 = Loop links

*Possible error codes:* -10 -40 -41 -43
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

# Part VI

# Obsolete functions

# 6 Obsolete functions

These functions are all obsolete and should be replaced by new functions as listed under each function:

They will all at some point be removed from RW Net.

The VRP functions has been marked as obsolete since the quality is not satisfying at the moment. They will later be replaced with other methods.

## 6.1 Coord2

Use Coord3 61 instead.

## 6.2 Direction0

### Function Direction0(x1,y1: double; node2: integer; x3,y3: double): integer;

Same as Direction2 67, but doesn't require file coord3.bin to be present. Instead direct access to polyline coordinates are required.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 6.3 FindNode

### Function FindNode(XLong,YLat: double; var node: integer; var dist: single): integer;

Obsolete function. Use Coordinate2Node 64 instead.

Finds the nearest node based on a geographic position. Returns node number and air distance in node and dist.

*Possible error codes:* -10
*ActiveX / VCL / CLX component:* RWnetBase

## 6.4 FindNodeFast

### Function FindNodeFast(XLong,YLat: double; var node: integer; var dist: single): integer;

Obsolete function. Use Coordinate2Node 64 instead.

Same as FindNode, except it uses the index created by NodeIndexCreate and is a lot faster.

When dealing with latitude/longitude coordinates either along the dateline or close to the poles, use FindNode instead. (Very unlikely to have a network there)

© 2013 RouteWare / Uffe Kousgaard

If the node index hasn't been created, the function will do so.

*Possible error codes:* -10 -34 -40 -41
*ActiveX / VCL / CLX component:* RWnetBase

## 6.5 IsoPoly1

### Function IsoPoly1(filename: string; node: integer; cost1,cost2: single; numpoly: integer; doughnut: boolean; convex: single): integer;

Obsolete function. Use Isopoly2 81 instead

IsoPoly1 is a wrapper around function IsoPoly1new 156 kept for backward compatibility. See this function for a description of parameters except these:

The value of cost1 is ignored.
Cost2 specifies the cost used at the end of the polygon.
Numpoly is the number of polygons created. Numpoly>=1.

Example: If numpoly is 3 and cost2 is 15: Isocost curves of 5, 10 and 15 are generated by IsoPoly1new.

*Possible error codes:* -10 -30 -32 -37 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 6.6 IsoPoly1new

### Function IsoPoly1new(filename: string; node: integer; doughnut: boolean; convex: single): integer;

Obsolete function. Use Isopoly2 81 instead. Only reason to use this function is if you are working with large networks, large isochrones and have little RAM or patience.

IsoPoly1new calculates isocost polygons, which shows how far it is possible to go within a specified amount of cost from a node.

The result is saved to filename, which should include path and filename, but exclude extension.

The output format is determined by property GISformat 74.

Node specifies the start-node.

Use procedure StepsClear 119 and StepsAdd 119 to add a list of the actual isochrones generated.

Doughnut defines if the polygons are created as hollow polygons (doughnut) or not.

The best way to understand these parameters is to try rwnetdemo.exe, which makes it possible to adjust all parameters easily and see the result immediately.

© 2013 RouteWare / Uffe Kousgaard

Convex defines to what degree the polygons should be calculated as convex hulls. The area around the possible points are scanned to see if other points are further away from the start-node. If not, the point is included in the polygon. Convex is specified in terms of degrees, when scanning. If convex is 0, only points on the convex hull is included. If the polygon is small, you would usually use a high value (30), whereas large polygons should be created with a smaller value (2-3).

If convex<>0 degenerate polygons may be created in some rare situations.

Latitude/longitude coordinates are not really supported by this function, but only in rare situations will it actually affect the generated polygons.

*Possible error codes:* -10 -30 -32 -37 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 6.7 NodeIndexCreate

### Function NodeIndexCreate: integer;

Creates an index on node-coordinates, which is needed by FindNodeFast and FindCloseNodes. Uses 4 bytes per node.

This is not needed anymore, since the spatial index has the same function.

*Possible error codes:* -10 -40 -41 -43
*ActiveX / VCL / CLX component:* RWnetBase

## 6.8 NodeIndexDrop

### Procedure NodeIndexDrop;

Frees memory associated by the node index.

This is not needed anymore, since the spatial index has the same function.

*ActiveX / VCL / CLX component:* RWnetBase

## 6.9 NodeIndexExist

### Function NodeIndexExist: boolean;

Returns true if the node index has been created.

This is not needed anymore, since the spatial index has the same function.

*ActiveX / VCL / CLX component:* RWnetBase

© 2013 RouteWare / Uffe Kousgaard

## 6.10 RouteVia

### Function RouteVia(Node1, Node2, Nodenum: integer): integer;

RouteVia calculates the cheapest path from node1 to node2 via one of the nodes entered into the standard nodelist 92.

Nodenum holds the number of nodes in the list.

The function makes sure that the result is returned in the fastest possible way for the loaded network. If oneway streets are used, it can not be avoided that the calculations may take some time.

Returns the id of the node in the nodelist.

*Possible error codes:* -10 -30 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 6.11 TSP

### Function TSP(nodenum: integer; TSPtype: smallint): single;

Obsolete function. Use TSP2 120 instead.

Calculates the optimum order to visit a list of nodes. TSP uses Lin's 3-optimum algorithm, which usually returns true optimum, if you have less than 20-30 nodes. With 100 nodes you should expect a cost no more than 1-2% higher than true optimum.

The cost between two nodes are calculated in both directions, but the average is used in the optimization.

Nodenum is the number of nodes in the optimization (2<=nodenum<=NodeListLimit 92).

TSPtype=0,10:
A route is calculated, which starts at the first node and ends at the last node in the list. The order of the other nodes are optimized.

TSPtype=1,11:
A round trip is calculated, where the order of all nodes are optimized.

TSPtype=2,12:
A trip is calculated, where only the first node is fixed. The last node on the route will be somewhere far from the first node. This method requires more computing time than type 0 and 1.

If TSPtype=10, 11 or 12 straight line distances are used in the optimization. This is a lot faster, but also less accurate.

The function returns the minimum cost.

© 2013 RouteWare / Uffe Kousgaard

At the same time the "extra" variable is calculated for the whole route. Get this with function TSPextra 159.

The new (optimum) order of the nodes can be retrieved with function NodeListGet 91.

If you enter a node number more than once, a short distance will be used for the distance between each occurrence.

With many nodes, the processing time increases a lot - especially for mode 0, 1 and 2.

Entering node number 0, triggers error -38 and so do nodes in a subnet.

*Possible error codes:* -10 -30 -38 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWcalc

## 6.12 TSPextra

### Function TSPextra: single;

This returns the extra variable after a call to function TSP 158, if extravar was defined in advance.

## 6.13 TurnExport

### Function TurnExport(filename: string): integer;

This function saves all present turn restrictions to a file on disk, which can later be loaded with function TurnImport 159.

If filename is empty the file's name is "turn.bin", If filename is "01", the file's name will be "turn01.bin" etc. This makes it possible to keep several sets of turn restrictions on disk for the same network.

The file will be saved in the same directory as specified by property Directory 67.

*Possible error codes:* -10 -18 -32 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 6.14 TurnImport

### Function TurnImport(filename: string; clear: boolean): integer;

This function loads turn restrictions from a file on disk, which was created by function TurnExport 159. If filename is empty the file's name is expected to be "turn.bin". If filename is "01", the file's name is expected to be "turn01.bin".

The file should reside in the same directory as specified by property Directory 67.

If clear is true, all present turn restrictions are deleted first from memory. It is not necessary to specify clear, if you have just loaded the network.

© 2013 RouteWare / Uffe Kousgaard

*Possible error codes:* -10 -18 -32 -40 -43
*Versions:* Standard Pro
*ActiveX / VCL / CLX component:* RWnetBase

## 6.15 VRP

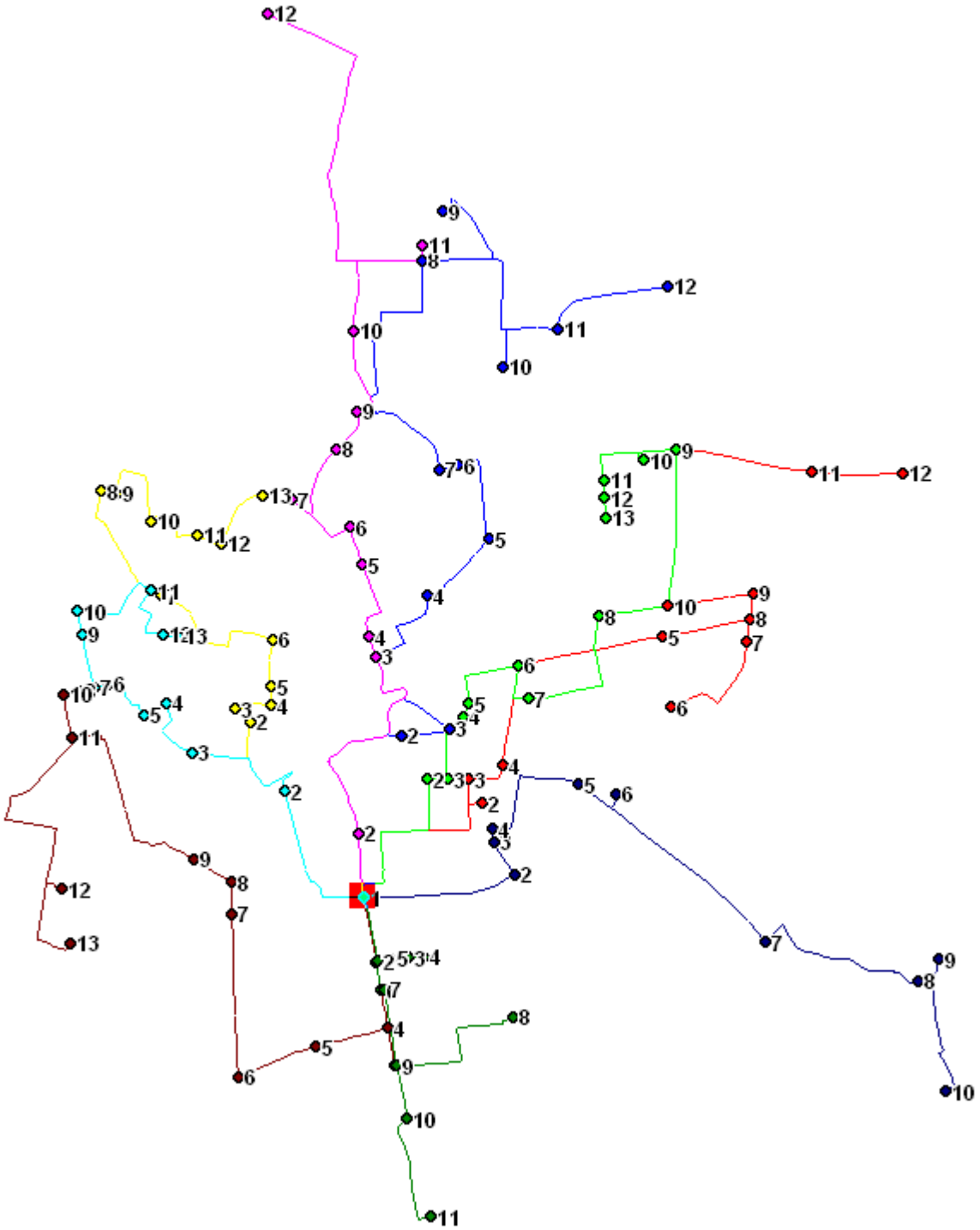
The VRP functions can help solving the basic multiple vehicle TSP problem, known as VRP. Problem is there are more customers to visit than a single vehicle can handle, so multiple routes has to be calculated. Below is listed the options available in this algorithm:

1. It is possible to define 2 load values for all customers. This can be number of passengers and weight for instance.
2. The maximum load is the same for all vehicles.
3. It is possible to define a service time for all customers (it takes X minutes to make a stop).
4. The maximum length (km or miles) and / or duration of the routes can be defined. This is the same for all vehicles.
5. It can be defined if the routes should be round-trips, in-bound or out-bound.

There are 3 algorithms available to choose between. Each of them may perform best for different kind of problems.

The map below show an example of the Sweep algorithm in out-bound mode. The red marker shows the depot:

© 2013 RouteWare / Uffe Kousgaard



This sample code is based on the standard demo data and generates the map seen above:

```
rwnetbase1.NWLoad
rwcalc1.ExtraVarCreate
rwcalc1.SetFastest
```

© 2013 RouteWare / Uffe Kousgaard

```
// 40 km/h on all links
for i = 1 to rwnetbase1.LinkMax
  rwnetbase1.SetLinkSpeed(i,40)

centernode = 1  // Node 1 = the depot
N = 100         // 100 customers, we use node 21-120 for ease
tripmode = 1    // tripmode 1 (out-bound)

rwcalc1.VRPcreate(N)

// setup coordinates
rwcalc1.VRPcoordinate(0,rwnetbase1.NodeCoordX
(centernode),rwnetbase1.NodeCoordY(centernode))
for i = 1 to N
  rwcalc1.VRPcoordinate(i,rwnetbase1.NodeCoordX(i
+20),rwnetbase1.NodeCoordY(i+20))

// demand1 = 1 person
// servicecost = 2 minutes to pick up each person
for i = 1 to N
  rwcalc1.VRPcustomer(i,1,0,2,0)

// maxload1 = 12 persons in each vehicle, max time = 35 minutes
rwcalc1.VRPvars(12,0,35,0)

// determine cost matrix
for i = 0 to N
begin
  if i=0 then startnode = centernode else startnode = i+20
  rwcalc1.IsoCost(startnode,0)
  for j = 0 to N
  begin
    if j=0 then endnode = centernode else endnode = j+20
    rwcalc1.VRPmatrix(i,j,rwcalc1.GetNodeCost(endnode),0)
  end
end

// Run model
rwcalc1.VRPrunSweep(tripmode)

// Get main results back
rwcalc1.VRPresults(routecount,cost,extra)

// create a GIS file showing all routes
RWcalc1.RouteFileCreate("route2",1)
RWcalc1.RouteFileFieldAdd("routeID",2,0,0)

for i = 1 to routecount
begin
  // get route-specific results back
  rwcalc1.VRPresultsRoute(i,customercount,cost,extra,load1,load2)
  for j = 1 to CustomerCount
  begin
    index1 = j
    index2 = j+1
    if j=CustomerCount then index2 = 1
    if index1=1 then
      node1 = centernode else
      node1 = rwcalc1.VRPresultsRouteIndex(i,index1)+20
    if index2=1 then
```

© 2013 RouteWare / Uffe Kousgaard

```
      node2 = centernode else
      node2 = rwcalc1.VRPresultsRouteIndex(i,index2)+20
    if  (tripmode=0) or
       ((tripmode=1) and (j<>CustomerCount)) or
       ((tripmode=2) and (j<>1)) then
    begin
      rwcalc1.Route(node1,node2)
      length = RWcalc1.RouteFind(node2)-1
      RWcalc1.RouteFileRecordAdd(length,inttostr(i))
      for k = 1 to length
        rwcalc1.RouteFileRecordAddSection(abs(rwcalc1.RouteGetLink
(k)),0,1)
    end
  end
end
rwcalc1.RouteFileClose
rwcalc1.VRPdrop
```

### 6.15.1 VRPcreate

## Function VRPcreate(size: integer): integer;

This function allocates RAM for solving a VRP instance with "size" places to visit. With 1 center (depot) and X places to visit, size=X. It returns 0 if no errors occured.

*Possible error codes:* -10 -30
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.2 VRPmatrix

## Function VRPmatrix(index1,index2: integer; cost,extra: single): integer;

Use this function to enter the cost and extra pairwise for all places to visit. Index1 >= 0, Index2 >= 0.

It returns 0 if no errors occured.

*Possible error codes:* -30 -57
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.3 VRPcustomer

## Function VRPcustomer(index: integer; demand1,demand2,servicecost,serviceextra: double): integer;

Use this function to enter the demand for all places to visit. The total demand for a route = load.

Servicecost or serviceextra can be time spent at the customer. If cost has been setup as time and you need 5 minutes to stop every place, use servicecost=5.

Index >0.

© 2013 RouteWare / Uffe Kousgaard

It returns 0 if no errors occured.

*Possible error codes:* -30 -57
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.4 VRPvars

## Function VRPvars(maxload1,maxload2,maxcost,maxextra: double): integer;

Use this function to enter parameters about the VRP to be solved.

Maxload1 and maxload2 can be used for defining maximum load(s) for all vehicles (*same* for all). Load can be weight, # of passengers or any other 2 measurements.

Maxcost and Maxextra can be used for defining a maximum length of the route as either time or distance, depending on how cost and extra has been defined.

At least one of these 4 parameters should be >0 or all customers will be assigned to a single route and then you have the normal TSP 120 problem.

It returns 0 if no errors occured.

*Possible error codes:* -30 -57
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.5 VRPcoordinate

## Function VRPcoordinate(index: integer; x,y: double): integer;

Use this function to enter coordinates for all places to visit.
Index 0 is the depot, the rest are the actual customers.

This is only required for method VRPrunSweep 164.
It returns 0 if no errors occured.

*Possible error codes:* -30 -57 -62
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.6 VRPrunSweep

## Function VRPrunSweep(tripmode: integer): integer;

Performs the actual solving of the VRP instance, using the classic sweep method. This method requires coordinates 164 to have been entered.

Tripmode can have 3 values:

0) Round-trip. All vehicles start at the depot and returns to it after having visited the last customer.

© 2013 RouteWare / Uffe Kousgaard

1) Out-bound: All vehicles start at the depot, but stops after having visited the last customer.

2) In-bound: All vehicles start at one of the customers and stops at the depot.

*Possible error codes:* -30 -57 -58
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.7 VRPrunCut

## Function VRPrunCut(tripmode: integer): integer;

Performs the actual solving of the VRP instance, using the Cut method. This is also known as route-first-cluster-second.

See VRPrunSweep 164 for an explanation of tripmode parameter.

*Possible error codes:* -30 -57 -58
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.8 VRPrunGrow

## Function VRPrunGrow(tripmode: integer; beta1,beta2: double; count: integer; var beta: double): integer;

Performs the actual solving of the VRP instance, using our own Grow method. It takes a parameter beta as input and it is possible to test with different values and get the best value returned:

This setup will for instance try with values 0.5, 0.6 and 0.7:
VRPrunGrow(tripmode, 0.5, 0.7, 3, beta);

See VRPrunSweep 164 for an explanation of tripmode parameter.

*Possible error codes:* -30 -57 -58
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.9 VRPresults

## Function VRPresults(var routecount: integer; var cost,extra: single): integer;

Return the number of routes generated and total cost and extra value. Use function VRPresultsRoute 166 for results on each route.

*Possible error codes:* -30 -57
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard

### 6.15.10 VRPresultsRoute

## Function VRPresultsRoute(route: integer; var customercount: integer; var cost,extra: single; var load1,load2: double): integer;

Return the number of customers, cost value, extra value, load1 and load2 on each route. Use function VRPresultsRouteIndex 166 for ordered list of customers on each route.

*Possible error codes:* -30 -57
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.11 VRPresultsRouteIndex

## Function VRPresultsRouteIndex(route, index: integer): integer;

Return the ordered index for each route.

*Possible error codes:* -30 -57
*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

### 6.15.12 VRPdrop

## Procedure VRPdrop;

This function de-allocates RAM for the VRP instance.

*Versions:* Pro
*ActiveX / VCL / CLX component:* RWcalc

© 2013 RouteWare / Uffe Kousgaard